Markt&Technik

Rudolf Schineis, Michael Braun, Thomas Grellner

C128 ROM-Listing:

BASIC 7.0-Betriebssystem

Ausführlich dokumentiertes Listing der Betriebssystem-Routinen. Mit detaillierter Beschreibung der Struktur und Arbeitsweise des BASIC-Interpreters.

Mit Crossreferenz-listen

C128 ROM-Listing: BASIC 7.0-Betriebssystem

Die Autoren:

RUDOLF SCHINEIS, geboren 1947 in Bayreuth, seit über 10 Jahren in der Ausbildung tätig, befaßte sich mit Mikrocomputern und Maschinenprogrammierung schon zu Zeiten des Ur-PET etwa 1977. Autor der Betriebssystem-Listings für die Commodore-Geräte CBM 8000, CBM 700, C64 und Floppy 8250.

MICHAEL BRAUN, geboren 1948 in Saarbrücken, unterrichtet seit 1976 Informatik in Berlin, auch ein Ur-PET-Maschinenprogrammierer. Autor der Betriebssystem-Listings für die Commodore-Geräte CBM 8000, CBM 700, C64 und Floppy 8250.

THOMAS GRELLNER, geboren 1964 in Bayreuth, Student der Betriebswirtschaft. Seit einigen Jahren Hobbyprogrammierer. Autor des Programms »Newtim 700«.

Nachdem Betriebssystem, Kernal-Editor und Monitor des C128 bereits im ersten Buch dieser Reihe (C128-ROM-Listing: Operating System, Bestellnummer MT 90221 im Markt&Technik Verlag) durch die Autoren ausführlich dokumentiert wurden, befaßt sich das vorliegende ROM-Listing mit dem BASIC 7.0 des C128. Es schließt damit die Dokumentation der ROM-Inhalte ab, so daß beide Bücher zusammen ein umfassendes und vollständiges Nachschlagewerk zum C128 bilden.

Zu Beginn des Buches wird der Leser in die Arbeitsweise des BASIC-Interpreters eingeführt, um ihm das Verständnis der ROM-Routinen zu erleichtern.

Themen sind unter anderem: Direktmodus und Programm-Modus, Aufbau des BASIC-Programmtextes, BASIC-Variablen und die Gleitkomma-Arithmetik. Auch die Implementation selbstgeschriebener BASIC-Erweiterungen wird an einem Beispiel demonstriert.

Der größte Teil des Buches wird vom ROM-Listing selbst eingenommen, angeführt von einer Beschreibung der Zero-Page, der erweiterten Zero-Page, der Memory-Management-Unit (MMU) und aller I/O-Bausteine wie VIC, SID, VDC, CIAs. Das ROM-Listing selbst erstreckt sich von hexadezimal $4000 bis $b000 und umfaßt damit 28 KByte.

Das Listing ist vollständig kommentiert und in Form eines Assembler-Quelltextes gehalten, eine Orientierung innerhalb der ROMs wird durch eine Crossreferenzliste erleichtert.

In diesem Buch finden Sie:

- Organisation und Arbeitsweise des C128-BASIC 7.0, Programmtext, Direkt- und Programm-Modus, Variablen, Arithmetik, Befehlserweiterungen
- Beschreibung von Zero-Page und erweiterter Zero-Page, I/O-Bausteinen, MMU
- ein vollständig dokumentiertes und kommentiertes ROM-Listing des BASIC 7.0 mit Crossreferenzliste einschließlich Markierung aller Definitionen und sich ändernder Speicherstellen.

Hardwareanforderungen:
Commodore 128 oder
Commodore 128 D

ISB N 3-89090-220-0

Markt&Technik

4 001057 902206

DM 49,-
sFr. 45,10
öS 382,20

Rudolf Schineis
Michael Braun
Thomas Grellner

C 128-ROM-Listing: BASIC-7.0-Betriebssystem

Ausführlich dokumentiertes Listing
der Betriebssystem-Routinen.
Mit detaillierter Beschreibung der
Struktur und Arbeitsweise
des BASIC-Interpreters

Markt&Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Schineis, Rudolf:
C 128, ROM-Listing: BASIC 7.0-Betriebssystem : ausführl. dokumentiertes Listing d. Betriebssystem-Routinen ; mit detaillierter Beschreibung d. Struktur u. Arbeitsweise d. BASIC-Interpreters / Rudolf Schineis ; Michael Braun ; Thomas Grellner. – Haar bei München : Markt-und-Technik-Verlag, 1986.
ISBN 3-89090-220-0
NE: Braun, Michael:; Grellner, Thomas:

Die Informationen im vorliegenden Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen.
Trotzdem können Fehler nicht vollständig ausgeschlossen werden.
Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Buch gezeigten Modelle und Arbeiten ist nicht zulässig.

»Commodore 128 Personal Computer« ist eine Produktbezeichnung der Commodore Büromaschinen GmbH, Frankfurt, die ebenso wie der Name »Commodore« Schutzrechte genießt.
Der Gebrauch bzw. die Verwendung bedarf der Erlaubnis der Schutzrechtsinhaberin.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
89 88 87 86

ISBN 3-89090-220-0

© 1986 by Markt&Technik, 8013 Haar bei München
Alle Rechte vorbehalten
Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Kösel, Kempten
Printed in Germany

INHALTSVERZEICHNIS

Vorwort		7
1	Der Interpreter (engl. Übersetzer)	9
1.1	Kaltstart des Interpreters	9
1.2	Warmstart des Interpreters	10
1.3	Die BASIC-Betriebs-Modi	10
1.3.1	Die Eingabeschleife - »Direkt-Modus«	11
1.3.2	Die CHRGET-Routine	11
1.3.3	Token	12
1.3.4	Umwandlung von BASIC-Befehlen in Token	12
1.3.5	Token-Tabelle	14
1.3.6	Direktausführungs-Modus - »Programm-Modus«	15
1.3.7	Programmier-Modus	16
1.3.8	Der BASIC-Programmspeicher	16
1.3.9	Internes Format einer BASIC-Zeile	17
1.4	Die Variablen	20
1.4.1	Regeln zur Benutzung von Variablennamen	21
1.4.2	Ablage der Variablen	21
1.4.3	Format für die Speicherung von Variablen	22
1.4.4	Stringvariablen	24
1.4.5	Integervariable	25
1.4.6	Gleitkommavariablen	26
1.4.7	Die Aufteilung des Variablenspeichers	29
1.5	Der Programm-Modus	30
1.6	Programmoptimierung	31
1.7	BASIC-Vektoren	32
1.8	Beispielprogramm	33
1.9	Tips und Tricks	37
1.9.1	Der »FRE()«-Befehl	37
1.9.2	Der »RENUMBER«-Befehl	38

1.9.3 Der »MID$«-Befehl 38
1.9.4 Das Horner-Schema 39

2 Grundsätzliches zum Assembler und den Assemblerlistings 41
2.1 Einführung in die Assemblerprogrammierung 41
2.2 Erklärung des Assemblerlistings 43
2.2.1 Zeilennummernfeld 43
2.2.2 Hex-Adressenfeld 43
2.2.3 Objekt-Code-Feld 44
2.2.4 Quell-Code-Feld 44
2.2.5 Kommentarfeld 46
2.3 Cross-Referenzliste beziehungsweise Verweistabelle 46

3 C-128-ROM-Listing: BASIC-Betriebssystem 49

4 Verweistabelle 531

Übersicht weiterer Markt&Technik-Produkte 588

VORWORT

Der »C 128« birgt im Vergleich zu den bisherigen Geräten ein neues, leistungsstarkes BASIC. Dieses »BASIC 7.0« ist Gegenstand des vorliegenden Buches. Im folgenden wird nicht die Sprache BASIC an sich, sondern das Maschinenprogramm BASIC aufgezeigt werden, denn die Programmiersprache BASIC kann nur durch ein vorhandenes Maschinenprogramm realisiert werden. Dieses Programm ist der hier geschilderte Interpreter.

Vorab noch einige Begriffserläuterungen. Im folgenden wird des öfteren von »RAM-Speicherseiten« die Rede sein. Dieser Begriff geht auf die interne Speicherverwaltung des »C 128« zurück. Der im »C 128« verwendete 8-Bit-Mikroprozessor »8502« ermöglicht lediglich einen Zugriff auf Speicherblöcke von maximal 64 KByte Größe. Ein solcher Speicherblock wird auch als »Bank« oder »Speicherseite« bezeichnet. Da der »C 128« jedoch mehr als 64 KByte Speicherplatz besitzt, wird der gesamte Speicher in Form von Speicherseiten zu je 64 KByte verwaltet. Die Bezeichnung »RAM« steht für »Random Access Memory«. Dies ist der Speicher, in dem Daten abgelegt und aus dem sie wieder gelesen werden können, solange der »C 128« in Betrieb ist. Die RAM-Speicherseiten werden numeriert und wie folgt verwendet:

RAM-0-Speicherseite - zur Speicherung von BASIC-Programmen
RAM-1-Speicherseite - zur Speicherung von BASIC-Variablen

Ebenso wird von Zeigern und Vektoren die Rede sein. Zeiger und Vektoren enthalten Adressen bestimmter Speicherstellen. Die Zeiger werden verwendet, um bestimmte Speicherinhalte mit Hilfe eines Indizierungsverfahrens zu lesen und zu überschreiben. Man unterscheidet im allgemeinen 1-Byte- und 2-Byte-Zeiger. Der Inhalt eines 1-Byte-Zeigers wird benötigt, um zu einer vorgegebenen Adresse einen sogenannten *Offset* zu verwalten. Dieser Offset gibt einen bestimmten Wert zwischen »$00« und »$FF« vor, der zu dieser Adresse addiert wird. Durch eine solche Adressierung von Speicherstellen werden Maschinenprogramme hinsichtlich ihres Platzbedarfes effizienter und die Programmierung selbst einfacher. Mit Hilfe eines

2-Byte-Zeigers kann der Mikroprozessor des »C 128« einen Adreßbereich von 64 KByte verwalten. Dies resultiert aus der Tatsache, daß ein Byte 256 Wertigkeiten (»$00« bis »$FF«) annehmen kann und mit zwei Byte 256 x 256 = 65536 Werte (»$0000« bis »$FFFF«) dargestellt werden können.

Vektoren werden als Sprungverteiler benötigt. Sie sind im Grunde 2-Byte-Zeiger, dienen jedoch einem anderen Zweck. Das heißt, man speichert in einem Vektor eine Adresse, an der ein Maschinenprogramm seine Ausführung fortsetzen soll, sobald eine sogenannte *indirekte Sprunganweisung* mit der Angabe dieses Vektors im Maschinenprogramm auftritt.

Zwei Bytes, die eine Adresse darstellen, werden jeweils gesondert benannt - man spricht von Low- und High-Bytes. Während das High-Byte den groben Teil der Adresse enthält und damit nur 256-Byte-Blöcke angibt, befindet sich im Low-Byte quasi die Feinstellung, d.h. der bytegenaue Adreßteil. Intern erfolgt die Speicherung einer Adresse in der Reihenfolge Low-Byte/High-Byte. Somit ergibt sich beispielsweise für die Hexadezimalzahl »$0100« mit dem High-Byte »$01« und dem Low-Byte »$00« das interne Format »$0001«. Die beiden Adreßbytes werden also in umgekehrter Reihenfolge gespeichert.

1 DER INTERPRETER (engl.: Übersetzer)

Der »C 128« verfügt über ein leistungsstarkes BASIC. Dieses »BASIC 7.0« ist wie seine Vorgänger (»BASIC 2.0«, »BASIC 4.0«) eine Interpreter-Sprache. Das bedeutet, daß ein eingegebenes BASIC-Programm sofort abgearbeitet werden kann, ohne es einer vorgeschalteten Übersetzung in einen für den Mikroprozessor verständlichen Code unterziehen zu müssen, wie dies bei sogenannten Compiler-Sprachen der Fall ist. Zudem ist es mittels eines Interpreters möglich, BASIC-Befehle direkt verarbeiten zu lassen. Gerade beim »C 128« ist dies entscheidend, da das BASIC hier gleichzeitig als Benutzeroberfläche fungiert, d.h., daß der »C 128« nach dem Einschalten sofort BASIC als Arbeitsprogramm zur Verfügung stellt.

1.1 Kaltstart des Interpreters

Der Interpreter des »C 128«, der sich in ROMs bzw. EPROMS befindet, ist ein sehr komplexes Maschinenprogramm, das bereits beim Einschalten des Computers aktiviert wird. Den Vorgang, der nach dem Einschalten bzw. nach dem Betätigen der »RESET«-Taste durchlaufen wird, nennt man Kaltstart. Dabei werden sämtliche intern vom Interpreter benötigten Variablen und Tabellen initialisiert. Spezielle Sprungvektoren, auf die im folgenden näher eingegangen werden wird, werden auf vorhandene Programmteile des Interpreters gerichtet. Diese Sprungvektoren dienen dazu, mögliche Erweiterungen des »BASIC 7.0« einfach und komfortabel in das vorhandene Interpreter-Programm des »BASIC 7.0« einzubinden, indem einfach diese Vektoren auf die eigenen Erweiterungen gerichtet werden und anschließend, nach der Behandlung der eigenen Erweiterung, die bisherigen Routinen angesprungen werden. Des weiteren werden Vektoren für ein BASIC-Programm und dessen verschiedene Variablentypen (String-, Integer- sowie Gleitkommavariablen) eingerichtet. Außerdem werden die Sound- und Graphikregister initialisiert. Danach erfolgt die Ausgabe der folgenden Einschaltmeldung am Bildschirm:

```
COMMODORE BASIC V7.0 122365 BYTES FREE
(C)1985 COMMODORE ELECTRONICS, LTD.
(C)1977 MICROSOFT CORP:
ALL RIGHTS RESERVED

READY.
```

Diese Einschaltmeldung gibt neben der BASIC-Version und den Copyright-Verhältnissen an, wieviel Speicher dem Benutzer im »BASIC 7.0«-Modus zur Verfügung steht. Die Angabe »122365 BYTES FREE« besagt, daß insgesamt 122365 Byte für ein BASIC-Programm und dessen Variablen verfügbar sind. Die Tatsache, daß diese Zahl nicht den Wert von 128 KByte wiedergibt, resultiert daraus, daß der »C 128« etwa 8700 Byte für seine interne Verwaltung benötigt.

Nachdem die Ausgabe der Einschaltmeldung erfolgt ist, versucht der »C 128«, das Betriebssystem CP/M 3.0+ von der Floppy zu »booten«, d.h. automatisch von einer Diskette zu laden. Ist der Ladevorgang erfolglos, weil beispielsweise keine CP/M-Diskette eingelegt ist, dann meldet der »C 128« sich mit »READY« zurück, er befindet sich nun im »BASIC 7.0«-Modus. Das heißt, der Interpreter kann nun benutzt werden.

1.2 Warmstart des Interpreters

Im Gegensatz zum Kaltstart kennt der Interpreter auch noch den sogenannten Warmstart. Dieser kann durch die Tastenkombination »STOP/RESTORE« (gleichzeitiges Drücken beider Tasten) jederzeit ausgelöst werden. Dabei wird quasi ein Einschalten des »C 128« simuliert, wobei die Sound- und Graphikregister sowie die BASIC-Vektoren zurückgesetzt werden. Der Versuch, CP/M 3.0+ zu »booten«, wird jedoch nicht unternommen. Ebenso wie beim Kaltstart meldet sich der »C 128« mit »READY.« betriebsbereit zurück. Ein vorher geladenes BASIC-Programm bleibt erhalten.

Mit der Meldung »READY.« gelangt das Interpreter-Programm sowohl nach dem Kaltstart als auch nach dem Warmstart in seine Hauptschleife. Diese Hauptschleife ist der wichtigste Teil des Interpreters. Hier werden alle Eingaben des Benutzers aufgefangen und verarbeitet.

1.3 Die BASIC-Betriebs-Modi

BASIC kennt zwei Betriebsarten. Den Direkt-Modus und den Programm-Modus. Als Direkt-Modus bezeichnet man die Betriebsart des BASIC, in der BASIC-Befehle direkt über die Tastatur eingegeben werden können und nach »RETURN« sofort durch den Interpreter des »C 128« ausgeführt

werden. Die Eingabe kann maximal 160 Zeichen groß sein, d.h. sie kann 2 Bildschirmzeilen im 80-Zeichen-Modus einnehmen.

Die andere Betriebsart wird Programm-Modus genannt. Hier werden den BASIC-Befehlen bei der Eingabe Zeilennummern vorangestellt. Der Interpreter erkennt daran, daß diese Programmzeilen nicht sofort abgearbeitet, sondern durch »RETURN« in den Programmspeicher übernommen werden sollen. Erst nach Eingabe des Befehls »RUN« und »RETURN« wird das eingegebene Programm ausgeführt.

Nach dem Einschalten des »C 128« gelangt der Rechner unmittelbar in den Direkt-Modus. Der »C 128« wartet dabei in einer Eingabeschleife auf eine Tastatureingabe.

1.3.1 Die Eingabeschleife - »Direkt-Modus«

Der Interpreter hat bei der Initialisierung einen Speicherbereich mit 160 Zeichen Länge in der RAM-0-Speicherseite als Eingabepuffer reserviert (die Adressen »$0200-$02A0«). Dieser Speicherbereich dient als Eingabepuffer, in dem die vom Benutzer eingetippten Zeichen zur Analyse und Umwandlung zwischengespeichert werden. Betätigt der Benutzer die »RETURN«-Taste, so wird die Eingabezeile vom Bildschirm in den Eingabepuffer kopiert, und die Auswertung der Eingabe seitens des Interpreters beginnt. Die eingetippten Zeichen liegen im Eingabepuffer als ASCII-Code vor. Das Ende der eingegebenen Zeichenkette wird mit dem ASCII-Code »NUL«=»$00« gekennzeichnet.

1.3.2 Die CHRGET-Routine

Der Inhalt des Eingabepuffers wird mit Hilfe der sogenannten CHRGET-Routine Zeichen für Zeichen vom Interpreter gelesen. CHRGET steht als Abkürzung für »Character Get« (Zeichen holen). Diese Routine, die bei der Initialisierung aus dem ROM ab der Adresse »$4279« in die RAM-0-Speicherseite nach »$0380« bis »$039E« kopiert wurde, liest mittels eines Zeigers in Adresse »$3D/$3E« (»$3D«=Low-Byte, »$3E«=High-Byte) ein Zeichen aus einer Speicherstelle des Rechners in der RAM-0-Speicherseite und übergibt dieses an die aufrufende Routine. Der Zeiger der CHRGET-Routine enthält während des Direkt-Modus einen Wert zwischen »$0200« und »$02a0«, d.h., in der Adresse »$3D« steht beispielsweise »$00« und in »$3E« steht »$02«. Dieser Wert entspricht der Anfangsadresse des Eingabepuffers. Es wird das erste Zeichen aus dem Eingabepuffer mittels der CHRGET-Routine gelesen und geprüft, um welche Art Zeichen es sich handelt. Wird der Wert Null übergeben, enthält der Eingabepuffer also keine Eingabe, beginnt die Eingabeschleife von neuem. Entspricht der übergebene Wert einer Ziffer (ASCII 0 bis 9), so werden die im Ein-

gabepuffer vorliegenden Zeichen in eine BASIC-Programmzeile umgewandelt, da es sich - wie an der führenden Zeilennummer zu sehen - um die Eingabe eines Programms handelt, welches erst im Programm-Modus abgearbeitet wird. Andernfalls versucht der Interpreter, die eingegebenen Zeichen als BASIC-Befehle direkt auszuführen. In beiden Fällen muß der Interpreter die im Eingabepuffer befindlichen Zeichen in sogenannte Token umsetzen.

1.3.3 Token

Ein Token besteht beim »BASIC 7.0« aus einem oder zwei Byte. Während vorhergehende BASIC-Versionen generell nur ein Byte verwendeten, werden hier, wegen der großen Anzahl der Befehle, auch Zwei-Byte-Token verwendet. Ein Token ist die Kurzform zur Speicherung von BASIC-Befehlen. Durch die Umwandlung von Befehlswörtern in Token werden zwei Effekte erzielt. Zum einen sind Token immer kürzer als die BASIC-Befehlswörter, und zum anderen verleihen sie BASIC-Programmen eine höhere Ausführungsgeschwindigkeit, da der Interpreter bei der Ausführung eines BASIC-Programms die Token als Index, ähnlich dem eines Buches, zum Auffinden der entsprechenden Programmteile benutzt.

Das erste Tokenbyte hat grundsätzlich einen Wert, der zwischen dezimal »128«, hexadezimal »$80«, und dezimal »254«, hexadezimal »$FE«, liegt.

1.3.4 Umwandlung von BASIC-Befehlen in Token

Die Umwandlung der BASIC-Befehlswörter in »Token« geschieht in der Routine »CRUNCH«, die im ROM ab der Adresse »$430D« beginnt. Hier wird zunächst der CHRGET-Zeiger (Adresse »$3D/$3E«) gesichert, da er im folgenden verändert werden wird.

»CRUNCH« untersucht nun den Eingabepuffer ab seiner Anfangsadresse »$0200«, bis die Zeilenendekennung »$00« gefunden wird. Zunächst wird eine eventuell voranstehende BASIC-Zeilennummer überlesen. Ist dies erfolgt, liest »CRUNCH« das nächste Zeichen aus dem Eingabepuffer. Handelt es sich dabei um die Eingabeendekennung »$00«, wird die Ausführung von »CRUNCH« beendet. Andernfalls wird geprüft, ob das gelesene Zeichen ein Doppelpunkt ist. Der Doppelpunkt fungiert im »BASIC 7.0« als Trennzeichen zwischen mehreren BASIC-Befehlen, die in einer Zeile eingegeben wurden. Tritt also ein Doppelpunkt auf, entspricht dies dem Ende eines BASIC-Befehls, woraufhin »CRUNCH« das nächste im Eingabepuffer befindliche Zeichen liest und dieses der oben genannten Überprüfung unterzieht. War das gelesene Zeichen kein Doppelpunkt, wird nun überprüft, ob es sich um ein Fragezeichen - die Abkürzung für den BASIC-Befehl »PRINT« - handelt. Liegt ein Fragezeichen vor, wird es

durch das dem BASIC-Befehl »PRINT« entsprechende Token »$99« ersetzt. Andernfalls vergleicht »CRUNCH« das gelesene Zeichen mit einem Anführungszeichen. Fällt der Vergleich positiv aus, werden alle folgenden Zeichen so lange überlesen, bis erneut ein Anführungszeichen oder eine Eingabeendekennung gefunden wird. Dies ist notwendig, da sonst Zeichen, die zur Ausgabe (etwa auf den Bildschirm) bestimmt sind, in Token verwandelt werden könnten. Eine gesonderte Behandlung kommt auch den beiden BASIC-Befehlswörtern »REM« und »DATA« zu. Der diesen Kommandos folgende Text wird ebenso überlesen und bleibt so, wie er eingetippt wurde, bestehen. Das gleiche gilt für Ziffern, so daß der BASIC-Befehl »GOSUB 1000« neben dem Byte für sein Token weitere 4 Byte für die Ziffernfolge »1000« in Anspruch nimmt. Erst wenn alle diese genannten Überprüfungen durchlaufen wurden, beginnt »CRUNCH« damit, die im ROM befindlichen Tabellen der BASIC-Befehlswörter mit den im Eingabepuffer befindlichen Zeichen zu vergleichen. Die Tabellen stehen im ROM ab der Adresse »$4417«. Stellt »CRUNCH« eine Übereinstimmung fest, wird der erste Buchstabe des im Eingabepuffer stehenden BASIC-Befehls durch das dem BASIC-Befehlswort entsprechende Token ersetzt. Der Rest des Eingabepuffers, der dem BASIC-Befehlswort folgt, wird dann direkt hinter das eingetragene Token kopiert. Danach fährt »CRUNCH« mit der Überprüfung des nächsten Zeichens fort. Wurde keine Übereinstimmung festgestellt, wird das im Eingabepuffer befindliche Zeichen überlesen und mit der Interpretation des nächsten Zeichens fortgefahren.

Der Interpreter hat im ROM drei Tabellen mit BASIC-Befehlswörtern gespeichert. Die erste Tabelle enthält die Standard-BASIC-Befehlswörter, die zum Teil schon vom »BASIC 4.0« bekannt sind. Die zweite Tabelle enthält die sogenannten erweiterten BASIC-Befehle, während die dritte die sogenannten erweiterten BASIC-Funktionen beinhaltet.

Der »C 128« kennt »169« BASIC-Befehlswörter. Da aber nur 127 Zeichen (»$80«-»$FE«) als Ein-Zeichen-Token zur Verfügung stehen, bedient sich der Interpreter eines zusätzlichen Vor- bzw. Pretokens. Das Pretoken hat den Wert »$CE« für die erweiterten BASIC-Funktionen und »$FE« für die erweiterten BASIC-Befehle. Dabei geschieht die Ermittlung des einzelnen Token auf sehr simple Weise. Bei den Standard-BASIC-Befehlswörtern wird zu Beginn des Vergleichsvorganges mit einer der Tabellen ein Zähler mit dem Wert 0 vorbesetzt und jeweils nach einem negativ verlaufenen Vergleich um 1 inkrementiert (erhöht). Bei einer Übereinstimmung wird zum Wert des Zählers »$80« addiert und das Resultat als Token verwendet. Bei den erweiterten BASIC-Befehlen und den erweiterten BASIC-Funktionen wird der besagte Zähler mit dem Wert 2 initialisiert und ebenfalls jeweils um 1 erhöht, wenn keine Übereinstimmung mit dem entsprechen-

den gespeicherten BASIC-Befehlswort erzielt werden konnte. Ist ein Vergleich positiv verlaufen, wird das entsprechende Pretoken, gefolgt von dem Wert des Zählers, in den Eingabepuffer kopiert.

Nachdem »CRUNCH« den Eingabepuffer in der beschriebenen Weise komprimiert und umgewandelt hat, liegt eine für den Interpreter ausführbare BASIC-Programmzeile vor. Eine Überprüfung des Inhalts der Zeilen auf Fehler findet jedoch erst bei der Ausführung selbst statt.

1.3.5 Token-Tabelle

Hier nun die Tabelle sämtlicher BASIC-Befehlswörter und deren Token:

Die Standard-BASIC-Befehle und ihre Token

end	$80	for	$81	next	$82
data	$83	input#	$84	input	$85
dim	$86	read	$87	let	$88
goto	$89	run	$8A	if	$8B
restore	$8C	gosub	$8D	return	$8E
rem	$8F	stop	$90	on	$91
wait	$92	load	$93	save	$94
verify	$95	def	$96	poke	$97
print#	$98	print	$99	cont	$9A
list	$9B	clr	$9C	cmd	$9D
sys	$9E	open	$9F	close	$A0
get	$A1	new	$A2	tab(	$A3
to	$A4	fn	$A5	spc(	$A6
then	$A7	not	$A8	step	$A9
+	$AA	-	$AB	*	$AC
/	$AD	↑	$AE	and	$AF
or	$B0	>	$B1	=	$B2
<	$B3	sgn	$B4	int	$B5
abs	$B6	usr	$B7	fre	$B8
pos	$B9	sqr	$BA	rnd	$BB
log	$BC	exp	$BD	cos	$BE
sin	$BF	tan	$C0	atn	$C1
peek	$C2	len	$C3	str	$C4
val	$C5	asc	$C6	chr$	$C7
left$	$C8	right$	$C9	mid$	$CA
go	$CB	rgr	$CC	rclr	$CD
joy	$CF	rdot	$D0	dec	$D1
hex$	$D2	err$	$D3	instr	$D4
else	$D5	resume	$D6	trap	$D7
tron	$D8	troff	$D9	sound	$DA

vol	$DB	auto	$DC	pudef	$DD
graphic	$DE	paint	$DF	char	$E0
box	$E1	circle	$E2	gshape	$E3
sshape	$E4	draw	$E5	locate	$E6
color	$E7	scnclr	$E8	scale	$E9
help	$EA	do	$EB	loop	$EC
exit	$ED	directory	$EE	dsave	$EF
dload	$F0	header	$F1	scratch	$F2
collect	$F3	copy	$F4	rename	$F5
backup	$F6	delete	$F7	renumber	$F8
key	$F9	monitor	$FA	using	$FB
until	$FC	while	$FD		

Die erweiterten BASIC-Befehle und ihre Token

bank	$FE $02	filter	$FE $03	play	$FE $04
tempo	$FE $05	movspr	$FE $06	sprite	$FE $07
sprcolor	$FE $08	rreg	$FE $09	envelope	$FE $0A
sleep	$FE $0B	catalog	$FE $0C	dopen	$FE $0D
append	$FE $0E	dclose	$FE $0F	bsave	$FE $10
bload	$FE $11	record	$FE $12	concat	$FE $13
dverify	$FE $14	dclear	$FE $15	sprsav	$FE $16
collision	$FE $17	begin	$FE $18	bend	$FE $19
window	$FE $1A	boot	$FE $1B	width	$FE $1C
sprdef	$FE $1D	quit	$FE $1E	stash	$FE $1F
fetch	$FE $21	swap	$FE $23	off	$FE $24
fast	$FE $25	slow	$FE $26		

Die erweiterten BASIC-Funktionen und ihre Token

pot	$CE $02	bump	$CE $03	pen	$CE $04
rsppos	$CE $05	rsprite	$CE $06	rspcolor	$CE $07
xor	$CE $08	rwindow	$CE $09	pointer	$CE $0A

1.3.6 Direktausführungs-Modus - »Programm-Modus«

Zurück zur Eingabeschleife. Wie oben ausgeführt wurde, kann es sich bei der eingegebenen Zeile sowohl um einen BASIC-Befehl als auch um mehrere direkt auszuführende BASIC-Befehle und auch um eine BASIC-Programmzeile handeln. Der Interpreter erkennt Programmeingaben an der der Zeile voranstehenden Zeilennummer.

Im ersten Fall wird nach dem Aufruf von »CRUNCH« das erste Zeichen aus dem Eingabepuffer erneut bereitgestellt und an die Routine

»XEQDIR« übergeben. »XEQDIR« steht für »direkt ausführen« und befindet sich ab der Adresse »$4AF3« im ROM des »C 128«. Mit Hilfe dieser Routine werden die eingegebenen BASIC-Befehle ausgeführt (siehe Programm-Modus).

1.3.7 Programmier-Modus

Im zweiten Fall wird die der Eingabezeile voranstehende BASIC-Zeilennummer in eine 2 Byte große Binärzahl mit einem Low- und einem High-Byte umgewandelt und zwischengespeichert. Dann erfolgt der Aufruf der Routine »CRUNCH«. Um die BASIC-Programmzeile an der richtigen Stelle im BASIC-Programmspeicher ablegen zu können, muß der Interpreter zunächst herausfinden, ob eine Programmzeile mit der gleichen Zeilennummer bereits vorhanden ist bzw. zwischen welchen existierenden Programmzeilen er diese einzufügen hat (Programmzeilen werden nach ihren Zeilennummern sortiert gespeichert). Dies wird mit der Routine »FNDLIN« (find line) realisiert, die ab der Adresse »§5064« im ROM steht. Gibt es im Speicher bereits eine Programmzeile mit der gleichen Zeilennummer, so wird diese gelöscht und an ihrer Stelle die neu eingegebene Programmzeile eingefügt. Andernfalls wird die Programmzeile zwischen zwei bestehende Zeilen, respektive am Beginn oder am Ende des BASIC-Programmspeichers, eingefügt.

1.3.8 Der BASIC-Programmspeicher

Der Anfang des BASIC-Programmspeichers liegt normalerweise bei »$1C00« der RAM-0-Speicherseite. Diese Adresse kann jedoch geändert werden, da sie über einen 2-Byte-Zeiger in der sogenannten Zero-Page verwaltet wird. Der Zeiger besteht aus dem Low-Byte in der Adresse »$2D« und dem High-Byte in »$2E«.

Das Ende des BASIC-Programmspeichers wird durch einen Zeiger in »$1212« und »$1213« festgelegt. Dieser Zeiger bezeichnet die Obergrenze innerhalb der RAM-0-Speicherseite, die ein BASIC-Programmtext nicht überschreiten kann. Im Gegensatz zum BASIC-Programmspeicherende kennt der »C 128« auch noch das BASIC-Textende, das ebenfalls durch einen Zeiger - in »$1210« und »$1211« - festgehalten wird. Dieser Zeiger weist nun allerdings auf das letzte Zeichen eines im Speicher befindlichen BASIC-Programmes hin.

Der Interpreter legt die BASIC-Programmzeilen im oben besprochenen Speicherbereich zwischen Textanfang und Textende in einem speziellen Ablageformat ab. Dieses Format ist notwendig, um eine höhere Verarbeitungsgeschwindigkeit sowohl beim Ablegen der Programmzeile als auch bei der Abarbeitung einer solchen zu erzielen.

Zunächst ist festzuhalten, daß ein BASIC-Programm nicht mehr Speicherplatz beanspruchen kann als ca. 58000 Zeichen. Dies begründet sich darin, daß für ein BASIC-Programm nur die RAM-0-Speicherseite mit einer Größe von 65536 Zeichen zur Verfügung steht und davon ca. 7500 Zeichen für die interne Verwaltung des »C 128« benötigt werden.

Durch die Limitierung des BASIC-Programmspeichers auf weniger als 64 KByte genügen somit 2 Byte, um die Adresse eines Bytes innerhalb dieses Speicherbereichs »$0000-$FFFF« (65535) darzustellen. Dies resultiert aus der Tatsache, daß ein Byte 256 Wertigkeiten annehmen kann und mit zwei Byte 256 x 256 = 65536 Werte dargestellt werden können. Deswegen liegt es nahe, auch die BASIC-Zeilennummern keine größeren Werte annehmen zu lassen, da sonst zum einen mehr Speicherplatz verbraucht würde und zum anderen sowieso nur maximal ca. »9600« BASIC-Programmzeilen gleichzeitig in den vorgegebenen Speicher passen. Diese Zahl ist jedoch rein hypothetisch, da von BASIC-Programmzeilen ausgegangen wird, die nur ein Token und keine anderen Zeichen enthalten. Die angesichts des doch sehr großen Speichers ziemlich gering erscheinende Anzahl von möglichen BASIC-Programmzeilen ergibt sich aus dem intern verwendeten Format, das im folgenden erläutert wird.

1.3.9 Internes Format einer BASIC-Zeile

Grundsätzlich wird eine BASIC-Programmzeile wie folgt abgelegt:

Am Anfang des BASIC-Programmspeichers, also Adresse »$1C00«, ist eine ASCII-»NUL« = »$00« eingetragen. Zu Beginn einer BASIC-Zeile steht eine 2 Bytes große Adresse (Low-Byte/High-Byte), die auf den Anfang der auf diese Zeile folgenden BASIC-Zeile zeigt. Diese Adresse wird auch als Link-Adresse bezeichnet (Link = Verbindung). Danach folgt die BASIC-Zeilennummer, ebenfalls in 2 Byte (Low-Byte/High-Byte) codiert. Ab der nächsten Speicherstelle wird die von der Routine »CRUNCH« mit Token versehene BASIC-Programmzeile gespeichert, wobei die als ASCII-Ziffernfolge vorliegende BASIC-Zeilennummer natürlich nicht mehr verwendet wird. Das Ende der so abgelegten BASIC-Programmzeile wird durch eine ASCII-»NUL« = »$00« gekennzeichnet. Die folgenden Programmzeilen werden im selben Format abgelegt. Das Ende eines BASIC-Programmes wird durch drei aufeinanderfolgende ASCII-»NUL«-Bytes kenntlich gemacht. Dabei handelt es sich bei der ersten ASCII-»NUL« um die reguläre Zeilenendekennung, bei den beiden folgenden um die Link-Adresse.

Für jede BASIC-Programmzeile gilt dieses Zeilenformat:

	1.+2. Byte Low/High	3.+4. Byte Low/High	5. bis n-tes Byte	letztes Byte
1	Link-Adresse	Zeilennummer	BASIC-Zeile	ASCII-»NUL«
2	Link-Adresse	Zeilennummer	BASIC-Zeile	ASCII-»NUL«
"	"	"	"	"
n	Link-Adresse	Zeilennummer	BASIC-Zeile	ASCII-»NUL«
n+1	2xASCII-»NUL«	>------- nicht benutzt		-------<

Die letzte Zeile der obenstehenden Tabelle zeigt die Programmendekennung, die der letzten (n-ten) BASIC-Zeile folgt.

Um dies an einem konkreten Beispiel sehen zu können, tippen Sie nach dem Einschalten folgende BASIC-Programmzeilen ein:

```
10 PRINT"ABCDE"          »RETURN«
20 PRINT"FGHIJ"          »RETURN«
```

Überträgt man dieses BASIC-Programm in das oben gezeigte Schema, erhält man folgende Tabelle:

	1.+2. Byte Low/High	3.+4. Byte Low/High	5. bis n-tes Byte	letztes Byte
1	0E 1C	0A 00	99 22 41 42 43 44 45 22	00
2	1B 1C	14 00	99 22 46 47 48 49 4A 22	00
3	00 00	>---------- nicht benutzt		----------<

Geben Sie nun den BASIC-Befehl »Monitor« ein:

```
MONITOR                  »RETURN«
```

Mit dem BASIC-Befehl »MONITOR« gelangt man in ein spezielles Programm des »C 128«, das es erlaubt, beliebige Speicherbereiche auf dem Bildschirm darzustellen. Von der Erklärung der einzelnen Befehle wird abgesehen (näheres siehe Handbuch).

Nachdem die Meldung des Monitorprogrammes erschienen ist, gibt man folgende Zeile ein:

```
M 01C00 01C1C          »RETURN«
```

Daraufhin erscheinen zwei Zeilen, jeweils beginnend mit der Adresse des angezeigten Speicherauszugs, gefolgt von »16« zweistelligen Hexadezimalzahlen und »16« ASCII-Zeichen, welche den ausgegebenen Zahlen entsprechen. Im folgenden nur die Abbildung des für dieses Beispiel wichtigen Teils der Bildschirmausgabe:

```
01C00 00 0E 1C 0A 00 99 22 41 42 43 44 45 22 00 1B 1C
01C10 14 00 99 22 46 47 48 49 4A 22 00 00 00
```

Die den beiden Zeilen voranstehenden fünfstelligen Hexadezimalzahlen geben die Adressen des jeweils ersten zweistelligen angezeigten Bytes der entsprechenden Zeile wieder. Die Adresse wird fünfstellig angezeigt, weil dadurch ersichtlich wird, aus welcher RAM-Speicherseite der Speicherauszug stammt. Die Kennung der RAM-Speicherseite ist durch die erste Ziffer der Adresse gegeben. In dem Beispiel handelt es sich um eine »0«. Das bedeutet, daß der Speicherauszug aus der RAM-0-Speicherseite, also dem BASIC-Programmspeicher, stammt.

In der Speicherstelle »$01C00« befindet sich die Anfangsmarke ASCII-NUL. Im Beispiel folgt die Link-Adresse zur nächsten BASIC-Programmzeile, die 2 Byte sind in der Form Low-Byte »$0E«, High-Byte »$1C« gespeichert. Dies ist auf eine Besonderheit des im »C 128« verwendeten Mikroprozessors zurückzuführen, der das höherwertige Byte einer Adresse immer an der höheren Adresse im Speicher erwartet. Die Adresse lautet also »$1C0E«. Sie gibt die Adresse des ersten Bytes der nächsten Link-Adresse wieder. Die auf die Link-Adresse folgende BASIC-Zeilennummer (low »$0A«, high »$00«) ist ebenfalls im vorgenannten Format abgelegt. »$00A0« ergibt dezimal geschrieben die Zahl »10«. Auf die BASIC-Zeilennummer folgt die von der Routine »CRUNCH« erzeugte komprimierte BASIC-Zeile (99 22 41 42 43 44 45 22), wobei »$99« das Token für den BASIC-Befehl »PRINT« wiedergibt und die folgenden sieben Zeichen die ASCII-Code-Folge für »"ABCDE"«. Direkt danach steht die Zeilenendekennung ASCII-NUL. Die nächste BASIC-Programmzeile (20 PRINT "FGHIJ") ist in derselben Weise abgelegt. Die Link-Adresse »$1B«»$1C« zeigt wiederum auf die nächste Programmzeile, und die Zeilennummer »$14«»$00« hat nun den Wert »20«. Danach folgt im Beispiel der Code »99 22 46 47 48 49 4a 22«, der den Inhalt der Programmzeile »20«, nämlich »PRINT"FGHIJ"«, wiedergibt. Die Link-Adresse (low »$1B«, high »$1C«) zeigt jetzt jedoch auf zwei ASCII-»NUL«-Bytes und damit auf die Programmendekennung. Weitere folgende Bytes sind deshalb für den Interpreter nicht von Interesse.

Um das Monitorprogramm zu verlassen, gibt man ein »x« für Exit ein und betätigt die »RETURN«-Taste.

Das soeben dargestellte Ablageformat für BASIC-Programmzeilen bringt erhebliche Geschwindigkeitsvorteile für den Interpreter. Dies wird beispielsweise in Verbindung mit der Routine »FNDLIN« sofort offensichtlich. Der Interpreter braucht nicht den ganzen BASIC-Programmtext Zeichen für Zeichen zu untersuchen, um eine bestimmte BASIC-Programmzeile zu finden. Vielmehr hangelt er sich von einer Link-Adresse zur nächsten und vergleicht lediglich die beiden einer Link-Adresse folgenden Bytes mit der gesuchten, ebenfalls in diesem Format vorliegenden BASIC-Zeilennummer.

1.4 Die Variablen

Eine Variable ist bekanntlich ein Platzhalter für einen Wert bzw. generell für Daten in einem BASIC-Programm. Variablen sind die einzige Möglichkeit unter BASIC, Daten speichern und verarbeiten zu können.

Das »BASIC 7.0« kennt drei Variablentypen zur Speicherung von Daten:

- Stringvariablen
- Integervariablen
- Gleitkommavariablen

Diese drei Typen können in Form von »einfachen« oder »indizierten« Variablen auftreten. Der Unterschied zwischen beiden Formen liegt darin, daß die indizierten Variablen, die auch als Feldvariablen bezeichnet werden, unter einem Namen eine vorher bestimmte Anzahl von Daten beinhalten können. Mit Hilfe des BASIC-Befehls »DIM« wird die besagte Anzahl festgelegt. Man unterscheidet bei den Feldvariablen ferner ein- und mehrdimensionale Feldvariablen. Eindimensional bedeutet, daß es sich bei der Variablen um eine einfache Liste von Dateneinträgen handelt. Mehrdimensionale Feldvariablen verdeutlicht man sich am besten, indem man sich eine oder mehrere Tabellen vorstellt, deren Einträge durch Angabe von Zeilen-, Spalten- und Tabellennummern bezeichnet sind.

Folgende Tabelle zeigt einige mögliche Verknüpfungen von Variablentypen und -versionen. Als Beispielvariable dient »X«:

Dimension	Gleitkomma	Integer	String
keine	X	X%	X$
eine	X(10)	X%(10)	X$(10)
zwei	X(10,20)	X%(10,20)	X$(10,20)
drei	X(2,15,30)	X%(2,15,30)	X$(2,15,30)

1.4.1 Regeln zur Benutzung von Variablennamen

- Das erste Zeichen des Variablennamens muß ein Buchstabe im Bereich von A-Z sein, also kein Umlaut oder »ß«;
- die folgenden Zeichen können auch Ziffern sein;
- es sind mehr als zwei Zeichen für den Variablennamen möglich, aber nur die ersten beiden Zeichen werden vom Interpreter als Name betrachtet;
- zur Unterscheidung des Variablentypes gilt:
  »%« hinter dem Variablennamen steht für Integertyp;
  »$« hinter dem Variablennamen steht für Stringtyp;
  folgt keines der beiden, handelt es sich um eine Gleitkommavariable;
- ein weiteres Folgezeichen kann »(« sein, um die Variable als Feldvariable zu kennzeichnen;
- schließlich darf der Variablenname kein BASIC-Befehlswort enthalten, da er sonst in ein Token umgewandelt würde.

Diese wenigen Regeln dienen der Schnelligkeit des Interpreters. Sie ersparen unnötig lange syntaktische Überprüfungen.

Besonders wichtig ist, daß bei der Verwendung von verschiedenen Variablen diese sich immer in den ersten beiden Zeichen unterscheiden müssen. Die Variablennamen »HAUS1« und »HAUS2« beispielsweise kennt der Interpreter nämlich nur als »HA« und damit nur als eine einzige Variable. Anders, wenn es sich bei »HAUS1« und »HAUS2« um verschiedene Variablentypen handelt. Etwa »HAUS1%« und »HAUS2$« werden intern als »HA%« und »HA$« erkannt und stellen daher zwei unterschiedliche Variablen dar.

1.4.2 Ablage der Variablen

Variablen benötigen Speicherplatz. Sie werden grundsätzlich in der RAM-1-Speicherseite gespeichert und verwaltet. Ebenso wie beim BASIC-Programmspeicher hält der Interpreter für die Variablenverwaltung einen Zeiger bereit, der auf den Anfang des Variablenspeichers zeigt: »$2F« = Low-Byte, $»30« = High-Byte.

Speziell für die Feldvariablen werden zwei weitere Zeiger vom Interpreter verwaltet. Zum einen gibt es einen Zeiger, der den Anfang der Feldvariablen und damit zugleich das Ende der einfachen Variablen beschreibt, und zum anderen einen Zeiger, der das Ende der Feldvariablen vermerkt.

Die Speicherung der Variablen erfolgt sequentiell, wobei der Speicher von unten nach oben gefüllt wird. Dabei werden jedoch die einfachen und die indizierten Variablen nicht willkürlich im Speicher plaziert, sondern im unteren Speicherbereich die einfachen und dahinter im oberen Speicherbereich die indizierten Variablen.

1.4.3 Format für die Speicherung von Variablen

Einfache Variablen. Für die einfachen Variablen gilt ein generelles Format bei allen drei Variablentypen. Alle pro Variable benötigten Informationen werden mit exakt 7 Byte beschrieben. Dabei geben die ersten beiden Bytes den Namen der Variablen wieder, die übrigen 5 enthalten die Daten, die in der Variablen gespeichert werden. Für die Darstellung des Namens einer Variablen werden zur Typenunterscheidung folgende Byte-Manipulationen vorgenommen:

Bei Stringvariablennamen wird der erste Buchstabe in seinem normalen ASCII-Code, der zweite Buchstabe mit »$80« ODER verknüpft dargestellt, das heißt, zu dem Byte wird 128 addiert. Der Variablenname einer Integervariablen wird kodiert, indem beide Buchstaben-Bytes mit »$80« ODER verknüpft werden. Bei Gleitkommavariablen werden beide Buchstaben in ihrem normalen ASCII-Code abgelegt. Hat ein Variablenname nur einen Buchstaben, wird das zweite Namensbyte als ASCII-NUL wiedergegeben.

Für die verschiedenen Variablentypen liegt folgendes Format vor:

Variablentyp	1. Byte	2. Byte	5. bis 7. Byte
Integer	Variablenname		Daten der Variablen
String	wie beschrieben		Daten der Variablen
Gleitkomma		in zwei Bytes	Daten der Variablen

Die folgenden Beispielvariablen »X«, »XA« und »X1« sollen dies veranschaulichen:

Variablentyp	1. Byte	2. Byte	5. bis 7. Byte
Integer	Variablenname		Daten der Variablen
X%	D8	80	Daten der Variablen
XA%	D8	C1	Daten der Variablen
X1%	D8	B1	Daten der Variablen
String	Variablenname		Daten der Variablen
X$	58	80	Daten der Variablen
XA$	58	C1	Daten der Variablen
X1$	58	B1	Daten der Variablen
Gleitkomma	Variablenname		Daten der Variablen
X	58	00	Daten der Variablen
XA	58	41	Daten der Variablen
X1	58	31	Daten der Variablen

Indizierte Variablen. Für indizierte Variablen wird ein flexibles Format verwendet. Ebenso wie bei den einfachen Variablen geben die ersten beiden Bytes in der oben genannten Typenkodierung den Namen des jeweiligen Feldes wieder. Danach folgen 2 Byte, welche die gesamte Länge des für das Feld benötigten Speicherplatzes enthalten. Das fünfte Byte gibt die Anzahl der Dimensionen an, über die sich die Feldvariable erstreckt. Es folgen ab der sechsten Byte-Position nun 2 Byte für jede Dimension. Diese Byte-Paare enthalten die Größe der jeweiligen Dimension plus 1, da Dimensionen nicht von 1 beginnend, sondern von 0 ab gezählt werden. Hatte eine hypothetische Feldvariable etwa drei Dimensionen, so lägen 6 beschreibende Byte vor.

Nach diesen, die Dimensionen einer Feldvariable beschreibenden Bytes folgen die Feldelemente. Um Speicherplatz zu sparen, werden die Daten jedoch nicht wie bei den einfachen Variablen in dem strikten 5-Byte-Format, sondern nur in der jeweiligen Typengröße abgelegt.

Das Format von Feldvariablen:

1.+2. Byte	3.+4. Byte	5. Byte	6.+7. Byte		n-tes Byte
Variablenname wie beschrieben	Länge der Feldvariablen low, high	Anzahl der Dimensionen	Letzte Dimension +1 low, high	...	Erste Dimension +1

Direkt dahinter folgen die Feldelemente:

Am Beispiel der eindimensionalen Variable »X(10)« und der zweidimensionalen Variable »X(10,10)« sei dies näher gezeigt:

1.+2. Byte	3.+4. Byte	5. Byte	6.+7. Byte	8.+9. Byte
X(10)				
58 00	3E 00	01	00 0B	---------> Daten
X(10,10)				
58 00	66 02	02	00 0B	00 0B :---> Daten

1.4.4 Stringvariablen

Unter dem Begriff »String« versteht man eine Folge von ASCII-Zeichen beliebiger Wertigkeit. In einer Stringvariablen sind also alle Zeichen denkbar, die der Computer kennt. Das sind insgesamt 256 Zeichen, bestehend aus Buchstaben, Ziffern, Satzzeichen, Währungssymbolen, Graphikzeichen und sogenannten Steuerzeichen.

Ein String kann maximal 255 Zeichen beinhalten. Diese Begrenzung wurde gewählt, da mit einem Byte maximal 256 Byte adressiert werden können. Liegt ein Zeiger auf den Anfang eines Strings vor, so kann der Mikroprozessor durch die Addition dieses Bytes zur Adresse des Zeigers ein bestimmtes Byte innerhalb des oben genannten Bereichs von 256 Zeichen lesen bzw. speichern. Es ist die schnellste Möglichkeit, um auf zusammenhängende Zeichenketten Zugriff zu erlangen.

Da Strings eine flexible Länge haben, können sie nicht in dem Standardformat von 5 Byte für einfache Variablen gespeichert werden. Der Interpreter verwendet in diesem Format auch nur 3 Byte zur Beschreibung eines Strings, den sogenannten String-Descriptor. Das erste Byte enthält dabei die Länge des Strings, die beiden anderen Bytes enthalten die String-Adresse. Die String-Adresse ist ein Zeiger, der die Adresse enthält, ab welcher der String sich im Speicher befindet.

Die Strings selbst, die Zeichenketten, werden in der RAM-1-Speicherseite von der höchsten verfügbaren Speicheradresse ab, zum Anfang der Speicherseite hin, abgelegt. Der abgespeicherte String wird wiederum von einem 2 Byte großen Zeiger gefolgt, der die Adresse des Bytes enthält, das die Länge des Strings wiedergibt. Liegt ein Feld aus Strings vor, so werden zur Beschreibung der Feldelemente jeweils nur 3 Byte benötigt.

Zur Verdeutlichung:

1.+2. Byte	3. Byte	4.+5. Byte	6.+7. Byte
Variablenname wie beschrieben	Länge des Strings	Stringzeiger	ungenutzt 2 x ASCII-»NUL«

Anhand der folgenden Beispielvariablen wird gezeigt, wie diese abgelegt werden.

X$ = "1"
XA$ = "2"
X1$ = "3"

1.+2. Byte	3. Byte	4.+5. Byte	6.+7. Byte
X$="1"			
58 80	01	FD FE	00 00
XA$="2"			
58 C1	01	FA FE	00 00
X1$="3"			
58 B1	01	F7 FE	00 00

1.4.5 Integervariable

Unter dem Begriff »Integer« versteht man eine Zahl ohne Nachkommastellen (eine ganze Zahl). Diese kann Werte im Bereich von -32768 und 32767 annehmen. Eine Integerzahl wird vom »C 128« intern in 2 Byte gespeichert, wobei das höchste Bit des High-Bytes das Vorzeichen der Zahl angibt. Ist dieses Bit gesetzt, also »1«, dann handelt es sich um eine negative Zahl, sonst um eine positive Zahl.

Die Integervariablen werden ebenfalls im 7-Byte-Format abgelegt. Dabei bleiben jedoch 3 Byte unbenutzt. Zur Verdeutlichung:

1.+2. Byte	3. Byte	4. Byte	5. bis 7. Byte
Variablenname wie beschrieben	Low-Byte der Integer-zahl	High-Byte der Integer-zahl	ungenutzt 3 x ASCII-»NUL«

Anhand der folgenden Beispielvariablen wird gezeigt, wie diese abgelegt werden.

X% = 1
XA% = 2
X1% = 3

1.+2. Byte	3. Byte	4. Byte	5. bis 7. Byte
X% = 1			
D8 80	00	01	00 00 00
XA% = 2			
D8 C1	00	02	00 00 00
X1% = 3			
D8 B1	00	03	00 00 00

Handelt es sich bei der Integervariablen um eine Feldvariable, wird kein Speicherplatz verschwendet, da in diesem Fall nur 2 Byte für die Speicherung jedes Integer-Feldelements verwendet werden.

1.4.6 Gleitkommavariablen

Gleitkommavariablen dienen zur Speicherung von Gleitkommazahlen, wie sie etwa bei Taschenrechnern üblich sind. Die interne Aufbewahrung von Gleitkommazahlen ist jedoch nicht unproblematisch. Dies wird deutlich, wenn man etwa eine Zahl wie 1/3 betrachtet. Die Zahl 1/3 ist kein endlicher Bruch, die Anzahl der Nachkommastellen kann im Computer jedoch nicht unendlich lang sein. Das BASIC des »C 128« gibt Gleitkommazahlen deshalb als Näherungswerte an, deren Verwendung logischerweise mit einem Verlust an Rechengenauigkeit verbunden sind. Um insbesondere nicht zuviel Speicherplatz zu verschwenden, werden Gleitkommazahlen in 5 Byte gespeichert. Dabei enthält das erste Byte einen Exponenten, der zur Basis 2 verwendet wird. Die übrigen 4 Byte enthalten die Mantisse. Im ersten Mantissen-Byte ist dabei das Vorzeichen der Gleitkommazahlen berücksichtigt. Da das Vorzeichen nur positiv oder negativ sein kann, genügt es, ein einzelnes Bit für seine Darstellung zu verwenden. Der »C 128« benützt dazu das höchstwertige Bit des ersten Mantissen-Bytes (1 = negativ, 0 = positiv). Damit hat der Rechner 4 Byte abzüglich 1 Bit, also 31 Bit zur Speicherung der Mantisse zur Verfügung.

Der Exponent enthält einen Wert, der eine Potenz zur Basis 2 wiedergibt, wobei zum Exponenten-Byte ein Offset von »128« addiert wird. Ein Wert von 131 im Exponenten-Byte entspricht dann einem Exponenten vom

Wert 3, ebenso wie einem Wert von 127 ein Exponent von -1 entspricht. Der Wert 0 im Exponenten-Byte bedeutet, daß die gespeicherte Gleitkommazahl den Wert 0 hat.

Das Vorzeichen-Bit im ersten Mantissen-Byte ist gesetzt, wenn die gespeicherte reelle Zahl negativ ist, andernfalls ist das Vorzeichen-Bit Null. Da das Vorzeichen-Bit das höchstwertige Bit in dem Mantisse-Byte ist, erhöht es, wenn es gesetzt ist, den Wert des Mantisse-Bytes scheinbar um 128 (2 hoch 7).

Die 4 Mantissen-Byte enthalten in der sogenannten normalisierten Darstellung einen Wert, der zwischen »0,5« und »1« liegt. Dieser Wert ergibt, multipliziert mit 2 hoch Exponenten-Byte minus 128, die angenäherte Gleitkommazahl.

Eine Gleitkommavariable wird wie folgt gespeichert:

1.+2. Byte	3. Byte	4. bis 7. Byte
Variablenname wie beschrieben	Exponent	Mantissen-Bytes 1 - 4

Anhand der folgenden Beispielvariablen wird gezeigt, wie diese abgelegt werden.

X = 1
XA = 2
X1 = 3

1.+2. Byte	3. Byte	4. bis 7. Byte
X = 1		
58 00	81	00 00 00 00
XA = 1		
58 41	82	00 00 00 00
X1 = 1		
58 31	82	40 00 00 00

Internes Gleitkommaformat. Das folgende Beispiel soll verdeutlichen, wie eine Gleitkommazahl in das sogenannte interne Format umgewandelt wird.

Betrachten wir als Beispiel die Zahl -13,2681: Das negative Vorzeichen setzt das höchstwertige Bit im ersten Mantissen-Byte. Der Exponent zur Basis 2 muß 4 betragen, denn 2 hoch 4 ist 16 und eine Mantisse zwischen 0,5 und 1 multipliziert mit 16 liegt im richtigen Zahlenbereich für unser Beispiel. Jetzt müssen wir noch die Mantisse bestimmen, die aus der Gleichung

$-13{,}2681 = \text{Mantisse} \times 2^4$

oder

Mantisse = -13,2681/16 = -0,82925625

Es bleibt nun die Aufgabe, die Mantisse -0,829... in 4 Mantissen-Byte aufzuteilen, das endgültige Exponenten-Byte in der Gleitkomma-Darstellung ergibt sich aus der Addition des Exponenten 4 und des Offsets 128 als 132 oder hexadezimal »$84«.

Die Wandlung der Mantisse erfolgt am besten in einem Rechenschema, das hier ohne eine Erläuterung der mathematischen Grundlagen zu Zahlenkonvertierungen einfach durchgeführt werden soll:

```
 0 , 82925625          Nachkommastellen jeweils x 16
-------------------------------------------------------
13 , 26810000
 4 , 28960002
 4 , 63360024
10 , 13760380
 2 , 20166016
 3 , 22656250
 3 , 62500000
10 , 00000000          usw.
```

Nimmt man nun die Vorkommastellen aus dem Rechenschema von oben nach unten

0, 13 4 4 10 2 3 3 10

und übersetzt die Zahlen in das hexadezimale Zahlensystem, erhält man

-0,8293 dezimal = -0,D4 4A 23 3A hexadezimal

Dabei entsprechen die vier hexadezimalen Ziffernpäckchen auf der rechten Seite der Gleichung gerade den vier Mantissen-Bytes, die wir benötigen.

Insgesamt erhalten wir für unser kleines Beispiel damit eine Gleitkomma-Darstellung von

-13,2681 --> 84 D4 4A 23 3A

Das Minuszeichen im ersten Mantissen-Byte ist nicht sichtbar, eine Eigenschaft der schon erwähnten normalisierten Darstellung, die sich dadurch

auszeichnet, daß innerhalb der Mantisse immer(!) das höchstwertige Bit gesetzt ist. Dies erreicht man, indem die Mantisse ggf. um Stellen nach links verschoben und der Exponent um 1 vermindert wird, beides hebt sich in der binären Darstellung gerade auf. Da in der normalisierten Darstellung also das erste Bit der Mantisse immer 1 ist, kann man in diesem Bit das Vorzeichen unterbringen. Bei Rechnungen, etwa in den Gleitkomma-Registern in der Zeropage, wird einfach das Vorzeichen aus der Mantisse herausgeholt und die 1 wieder eingefügt. Diese Vorgehensweise spart bei der Abspeicherung einer Gleitkommazahl, z.B. in einer Variablen, ein ganzes Byte, das sonst für das Vorzeichen benötigt würde.

1.4.7 Die Aufteilung des Variablenspeichers

Der Variablenspeicher des »BASIC 7.0« befindet sich in der RAM-0-Speicherseite. Er ist wie folgt aufgeteilt:

$1ffff - §1ff00	vom Betriebssystem genutzt
$1ffef - $10400	für die Variablenverwaltung und -speicherung verwendeter Speicherbereich
$103ff - $10000	vom Betriebssystem genutzt

Der Speicherbereich zwischen »$10400« und »$1ff00« wird variabel durch die zur Verwaltung des Variablenspeichers bestimmten Zeiger aufgeteilt. Die folgenden Zeiger werden, wie bereits bekannt, in der Reihenfolge Low-Byte, High-Byte dargestellt.

Zeiger	Beschreibung
$2f/$30	Anfang der einfachen BASIC-Variablen. Die Ablage geschieht in der Reihenfolge der Initialisierung der Variablen, wobei die drei möglichen Typen nicht sortiert abgelegt werden.
$31/$32	Ende der einfachen BASIC-Variablen und Beginn der indizierten BASIC-Variablen (sonst wie einfache Variablen)
$33/$34	Ende der indizierten Variablen
$35/$36	Anfang der Stringdaten, die vom oberen Speicherende zum unteren Speicherende abgelegt werden.
$37/$38	Ende der Stringdaten

1.5 Der Programm-Modus

Der Programm-Modus kann durch die BASIC-Befehle »RUN« (mit oder ohne Zeilennummer) sowie durch »GO TO« oder »GO SUB« (jeweils gefolgt von einer Zeilennummer) aus dem Direkt-Modus eingeleitet werden. Man spricht ebenfalls vom Programm-Modus, wenn ein BASIC-Befehl oder eine Aneinanderreihung von BASIC-Befehlen im Direkt-Modus eingegeben worden sind und diese dann ausgeführt werden. Der Interpreter verarbeitet dabei in allen Fällen durch die Routine »CRUNCH« vorher komprimierte BASIC-Befehlszeilen.

Grundsätzlich beginnt die Ausführung eines BASIC-Programms mit der Abarbeitung der im Eingabepuffer befindlichen Befehle zum Aufruf eines BASIC-Programmes. Dabei bleiben initialisierte Variablen bestehen. Lediglich wenn der Programmaufruf mit Hilfe des BASIC-Befehls »RUN« geschieht, werden die Variablen *gelöscht*. Das Löschen der Variablen geschieht nicht physikalisch, beispielsweise durch Überschreiben mit Nullbytes, die Variablen werden vielmehr einfach ungültig erklärt. Dies geschieht durch Zurücksetzen der Variablenzeiger. Zurücksetzen bedeutet, daß die Zeiger diejenigen Werte erhalten, die auch direkt nach dem Einschalten des Rechners in ihnen vorlagen. Da somit Anfangs- und Endezeiger der Variablen (siehe oben) auf die gleichen Adressen verweisen, erkennt der Interpreter, daß keine Variablen existieren.

Ein BASIC-Programm kann grundsätzlich nicht vom Mikroprozessor direkt ausgeführt werden. Vielmehr wird der komprimierte BASIC-Programmtext durch den Interpreter Zeichen für Zeichen gelesen und von ihm ausgeführt. Gelesen wird der Programmtext mit Hilfe der »CHRGET«-Routine. Der Zeiger der »CHRGET«-Routine enthält dabei eine Adresse, die entweder in den Eingabepuffer zeigt, falls ein BASIC-Befehl im Direkt-Modus eingegeben wurde, oder die Adresse gibt eine Speicherstelle des BASIC-Programmspeichers wieder. Am Anfang einer Befehlszeile muß ein Token stehen, das heißt ein kodierter BASIC-Befehl. Mit diesem Wert als Index wird eine Adresse aus einer Adreßtabelle gelesen. Die gelesene Adresse ist die Anfangsadresse einer Routine, die den BASIC-Befehl, für welchen das Token steht, verarbeitet. So ist jedem Befehl des »C 128« eine Routine des »BASIC 7.0« zugeordnet, die in Maschinensprache geschrieben ist und vom Mikroprozessor des Computers ausgeführt wird.

Wird ein BASIC-Programm durch einen der drei Befehle »RUN«, »GO TO« oder »GOSUB« gestartet, wird zunächst die dem jeweiligen Befehl entsprechende Routine des »BASIC 7.0« aufgerufen. In allen drei Fällen wird untersucht, ob dem Befehl-Token eine Zeilennummer folgt. Wird bei den Befehlen »GO TO« oder »GOSUB« keine Zeilennummer erkannt, erfolgt eine Fehlermeldung. Der »RUN«-Befehl kann auch ohne Zeilennum-

mernangabe erfolgen, dann wird die erste vorhandene BASIC-Programmzeile als auszuführende Zeile angenommen.

Allen drei Befehlen ist wiederum gemeinsam, daß sie einen Zeiger auf die abzuarbeitende BASIC-Befehlszeile liefern. Der Interpreter greift diese Adresse auf und setzt die Verarbeitung des BASIC-Programmes ab dieser Adresse fort.

Die einzelnen BASIC-Befehlsroutinen des Interpreters überprüfen selbst, ob die dem Token folgenden Zeichen syntaktisch korrekt sind. Die entsprechende Fehlermeldung »?syntax error« wird gegebenenfalls erzeugt und der Programmablauf abgebrochen.

Die für die meisten BASIC-Befehle notwendigen Parameter können wie folgt zusammengefaßt werden:

- Ziffernfolgen
- Strings in Anführungszeichen
- Variable
- BASIC-Befehle, die wiederum ein Ergebnis liefern

1.6 Programmoptimierung

Um schnelle BASIC-Programme schreiben zu können, sollte man die Arbeitsweise des Interpreters und einzelne seiner Routinen etwas näher kennen (siehe BASIC-ROM-Listing).

Zunächst gilt es zu beachten, daß Sprungbefehle wie beispielsweise »GO TO« immer zu einer Suche des Interpreters nach der anzuspringenden Zeile führen. Dies kostet speziell dann Zeit, wenn die gesuchte BASIC-Zeile weit hinter dem Programmanfang - obwohl vielleicht ganz in der Nähe der aktuellen BASIC-Zeile - liegt. Es empfiehlt sich also, so wenig wie möglich mit Sprungbefehlen zu arbeiten.

Ein weit größeres Zeitproblem stellt sich im Zusammenhang mit der Verwendung von Daten. Zunächst sollten alle im Programm verwendeten Variablen am Anfang des BASIC-Programmes initialisiert bzw. mit einem Wert belegt werden. Dadurch wird während des Programmlaufs keine Zeit mehr für unnötige Speicherverschiebungen im Variablenspeicherbereich verschwendet. Außerdem empfiehlt es sich, häufig gebrauchte Variablen vor allen anderen zu definieren, da die Variablen in einer Liste gespeichert werden, die jedesmal, wenn eine Variable auftritt, von Anfang an durchsucht werden muß. Des weiteren ist es von Vorteil, so wenig wie möglich Zahlen im Programmtext zu verwenden. Statt beispielsweise B = B + 1 zu schreiben, sollte man besser B = B + L verwenden, wobei »L = 1« vorher

definiert wurde, da Zahlen im BASIC-Programm als ASCII-Zeichen vorliegen und jedesmal erneut in ein internes Format umgewandelt werden müssen. Der erreichte Zeitvorteil beträgt zirka 15 Prozent.

1.7 BASIC-Vektoren

Das »BASIC 7.0« bietet viele Erweiterungsmöglichkeiten. So kann man einen Großteil der Interpreterfunktionen *anzapfen*. Das wird dadurch ermöglicht, daß für viele wichtige Routinen sogenannte Vektoren vorliegen. Die Vektoren enthalten, wie die schon mehrfach erwähnten Zeiger, Adressen. Nach dem Einschalten des Rechners werden diese Vektoren mit Standardadressen belegt, die auf vorhandene Routinen des Interpreters zeigen.

Als Beispiel wäre der Vektor, der auf die Fehlerausgabe-Routine »$4D3F« des »BASIC 7.0« zeigt, zu nennen. Der zugehörige Vektor »$0300/$0301« könnte beispielsweise auf eine vom Benutzer geschriebene modifizierte Fehlerausgabe-Routine zeigen, die statt der englischen deutsche Fehlermeldungen produziert. Auch auf die Eingabeschleife des Interpreters sowie die Routinen »CRUNCH«, »XEQDIR« und die BASIC-List-Routine ist jeweils ein Vektor gerichtet.

Hier eine Zusammenstellung der wichtigsten Vektoren:

Vektor Low-/High-Byte	Beschreibung
$02FC/02FD	Aufruf einer User-definierten Funktion
$0300/0301	Fehler-Routine für BASIC-Fehlermeldungen
$0302/0303	BASIC-Warteschleife bzw. Eingabeschleife
$0304/0305	Standardeinsprung in die »CRUNCH«-Routine
$0306/0307	Standardeinsprung in die »LIST«-Routine
$0308/0309	Standardeinsprung zur Programmverarbeitung
$030A/030B	Auswertung von beliebigen Ausdrücken
$030C/030D	BASIC-Erweiterung der »CRUNCH«-Routine
$030E/030F	BASIC-Erweiterung der »LIST«-Routine
$0310/0311	BASIC-Erweiterung zur Programmverarbeitung

BASIC-Erweiterungen. Mit Hilfe der vorstehenden Vektoren ist es unter anderem möglich, selbstprogrammierte BASIC-Befehle in das »BASIC 7.0« einzubinden. Dafür müssen eine eigene Routine zur Token-Umwandlung sowie eigene Programme, die dann die jeweiligen BASIC-Befehle ausführen, in Maschinensprache geschrieben werden. Ferner wird noch eine Tabelle mit den benutzerdefinierten BASIC-Befehlswörtern benötigt.

Will man eine BASIC-Erweiterung programmieren, empfiehlt es sich, dabei den Programmierstil, der für das BASIC-ROM verwendet wurde, nachzuahmen. Dies erleichtert und verkürzt die aufwendige Maschinenprogrammierung. Die entsprechenden Routinen findet man im BASIC-ROM-Listing.

Wichtig ist, daß die Werte der »Token« nicht mit denen von vorhandenen »Token« kollidieren. Die Erweiterung des »BASIC 7.0« über die oben genannten Vektoren ist auf die Verwendung der 2-Byte-Token gemünzt, so daß zum Beispiel für die erweiterten BASIC-Befehle des »BASIC 7.0« die Token »$02« bis »$27« nicht verwendet werden können. Im nachfolgend stehenden Beispielprogramm wird deshalb ein zweites Tokenbyte von »$30« gewählt.

1.8 Beispielprogramm

Anhand des folgenden Beispiels sei eine BASIC-Erweiterung dargestellt:

Es soll der neue BASIC-Befehl »OLD« implementiert werden.

Der »OLD«-Befehl. Der »OLD«-Befehl soll als Gegenstück zum »NEW«-Befehl des »BASIC 7.0« dienen. Mit ihm ist es möglich, ein BASIC-Programm, das mit dem BASIC-Befehl »NEW« gelöscht wurde, wiederherzustellen. Dazu muß man wissen, daß ein im Speicher befindliches BASIC-Programm durch »NEW« nicht wirklich gelöscht wird, sondern wie bei den Variablen lediglich Zeiger zurückgesetzt werden. Diese Zeiger gilt es durch »OLD« zu rekonstruieren. Wie oben bereits erwähnt, wird der BASIC-Programmspeicher hauptsächlich durch zwei Zeiger festgelegt. Dies sind der BASIC-Anfangzeiger »$2d/$2e« und der BASIC-Endezeiger »$1210/$1211«. Wurde der BASIC-Befehl »NEW« eingegeben, wird der BASIC-Endezeiger auf den Wert des BASIC-Anfangzeigers gesetzt. Außerdem wird die erste Link-Adresse des BASIC-Programmes mit Nullen geladen, so daß sie das BASIC-Programmende markiert.

Der »OLD«-Befehl macht diese Änderungen nun rückgängig. Dazu muß zunächst die erste Link-Adresse wiederhergestellt werden. Dies wird erreicht, indem ab dem ersten BASIC-Text-Byte nach einer ASCII-NUL gesucht wird, die das BASIC-Zeilenende markiert. Die Adresse des dar-

auffolgenden Bytes wird dann als erste Link-Adresse im Programm eingetragen. In der zweiten Phase wird das ursprüngliche Programmende gesucht, das durch drei aufeinanderfolgende ASCII-NUL-Bytes gekennzeichnet ist.

Zunächst sollen die BASIC-Vektoren für die BASIC-Erweiterungen gesetzt werden. Dazu benötigt man im allgemeinen kein Maschinenprogramm, da man über einen BASIC-Loader (ein BASIC-Programm, das ein Maschinenprogramm in den Speicher »poked« bzw. schreibt) sehr komfortabel die entsprechenden Speicherstellen ändern kann. Es wird hier jedoch der Vollständigkeit halber ein Maschinenprogramm gezeigt, das die Initialisierung der BASIC-Vektoren vornimmt:

Die OLD-Routine

```
00000 ===> Variablendefinition <===
00000
00000 ===> Zeiger <===
00000
00000           txttab = $2d            addrl Zeiger Anfang BASIC-Text
00000           iesclk = $030c          ind. Vektor CRUNCH-Erweiterung
00000           iescpr = $030e          ind. Vektor LIST-Erweiterung
00000           iescex = $0310          ind. Vektor Ausführungs-Routine
00000           fbufpt = $72            addrl Zeiger im Puffer
00000           txttop = $1210          addrl Zeiger Ende BASIC-Text

00000 ===> Unterprogramme aus dem BASIC 7.0 <===
00000
00000           chrget = $0380          BASIC-Text lesen
00000           nesclk = $4321          Token-Wandlung
00000           crun95 = $43b2          Token-Auswertung beenden
00000           reser  = $43e2          Befehlswort suchen
00000           p1l026 = $516a          Befehlswort listen
00000 ===> Konstante <===
00000
00000           eom    = $00            end-of-message
00000                * = $1300          Startadresse des Maschinenprogramms
00000

01300 ===> Initialisierung der BASIC-Vektoren <===
01300
01300 a9 20     start  lda #<mycrun     Adresse der Erweiterung der
01302 8d 0c 03         sta iesclk       CRUNCH-Routine in den iesclk-Vektor
01305 a9 13            lda #<mycrun     schreiben
01307 8d 0d 03         sta iesclk+1
0130a a9 40            lda #<mylist     Adresse der Erweiterung der
0130c 8d 0e 03         sta iescpr       LIST-Routine in den iescpr-Vektor
0130f a9 13            lda #<mylist     schreiben
01311 8d 0f 03         sta iescpr+1
01314 a9 50            lda #<myexec     Adresse der Erweiterung der
01316 8d 10 03         sta iescex       Ausführungs-Routine in den
01319 a9 13            lda #<myexec     iescex-Vektor schreiben
0131b 8d 11 03         sta iescex+1
0131e 60               rts              Rücksprung zur aufrufenden Routine
```

```
01320 ===> eigene Routine, um erweiterte BASIC-Befehle zu <===
01320              suchen und in Token zu wandeln
01320
01320 48        mycrun pha              ac sichern
01321 a9 13            lda #>cmdlst     Adresse der Befehlswörterliste
01323 a0 a0            ldy #>cmdlst     in ac high, in yr low an die
01325 20 e2 43         jsr reser        Befehlswort-Suchroutine übergeben,
01328 90 0d            bcc notfnd       falls nicht gefunden, -> notfnd
0132a aa               tax              ac sichern
0132b 68               pla              Stack vom alten ac bereinigen
0132c 8a               txa              und neuen ac zurückholen
0132d 29 7f            and #$7f         höchstes Bit ausmaskieren
0132f 09 30            ora #$30         und 48 odieren bzw. addieren
01331 a2 00            ldx #$00         Offset für erweiterten BASIC-Befehl
01333 4c b2 43         jmp crun95       Token-Auswertung beenden
01336 38        notfnd sec              C-Flag für normale Tokenisierung
01337 68               pla              Stack vom alten ac bereinigen
01338 4c 21 43         jmp nesclk       zurück zur normalen Token-Routine

01340 ===> eigene Routine, um erweiterte BASIC-Befehlswörter <====
01340           der LIST-Routine zur Verfügung zu stellen
01340
01340 29 0f     mylist and #$0f         4 High-Bits des Token ausmaskieren, da weniger
                                        als 16 Befehle vorhanden
01342 09 80            ora #$80         und höchstens Bit für LIST setzen
01344 aa               tax              zur Übergabe in xr geben
01345 a9 13            lda #>cmdlst     Adresse der Befehlswörterliste
01347 a0 a0            ldy #>cmdlst     in ac high, in yr low übergeben
01349 4c 6a 51         jmp p1l026       BASIC-Wort suchen und ausgeben

01350 ===> eigene Routine, um erweiterte BASIC-Befehle     <===
01350                zur Ausführung zu bringen
01350
01350 29 0f     myexec and #$0f         4 High-Bits des Tokens ausmaskieren, da
                                        weniger als 16 Befehle vorhanden
01352 0a               asl a            Inhalt des ac mit 2 multiplizieren
01353 a8               tay              ac für indirekten Zugriff -> yr
01354 b9 b1 13         lda adrlst+1,y   High-Byte der Adresse der Befehls-
01357 48               pha              Routine auf Stack legen
01358 b9 b0 13         lda adrlst,y     Low-Byte der Adresse der Befehls-
0135b 48               pha              Routine auf Stack legen
0135c 4c 80 03         jmp chrget       Aufruf der Befehls-Routine durch rts der
                                        chrget-Routine

01360 ===> OLD-Routine zur Wiedererlangung eines mit NEW <===
01360             gelöschten BASIC-Programmes
01360
01360 a5 2d     oldcmd lda txttab       BASIC-Textzeiger laden
01362 a4 2e            ldy txttab+1
01364 85 72            sta fbufpt       und als Hilfszeiger speichern
01366 84 73            sty fbufpt+1
01368 a0 03            ldy #$03         yr als Schleifenzähler initialisieren, Zeiger
                                        auf erstes Byte nach der BASIC-Zeilennummer
0136a c8         loop1 iny
0136b b1 72            lda (fbufpt),y    Zeichen aus BASIC-Textspeicher lesen
0136d d0 fb            bne loop1         wenn <>0 dann -> loop1
```

```
0136f c8              iny             yr zeigt jetzt auf zweite Link-Adresse im
                                      BASIC-Programm
01370 98              tya             yr für Berechnung nach ac
01371 18              clc
01372 65 72           add fbufpt      ac zur Anfangsadresse des BASIC-
01374 a0 00           ldy #$00        Programms addieren
01376 91 2d           sta (txttab),y  und in die erste Link-Adresse speichern
01378 a5 73           lda fbufpt+1
0137a 69 00           adc #$00        eventuellen Übertrag berücksichtigen
0137c c8              iny             und High-Byte ebenfalls speichern
0137d 91 2d           sta (txttab),y
0137f 88              dey             yr = 0
01380 a2 03     loop2 ldx #$03        xr als Schleifenzähler initialisieren
01382 e6 72     nxtnul inc fbufpt     Hilfszeiger erhöhen
01384 d0 02           bne notzro      wenn <>0 -> notzro
01386 e6 73           inc fbufpt+1    sonst Übertrag berücksichtigen
01388 b1 72     notzro lda (fbufpt),y Zeichen aus BASIC-Textspeicher lesen
0138a d0 f4           bne loop2       wenn <>0 dann -> loop2
0138c ca              dex
0138d d0 f3           bne nxtnul      xr <> 0 und ax = 0, dann -> nxtnul
0138f a5 72           lda fbufpt      drei aufeinanderfolgende Nullen gefunden und
                                      BASIC-Programmende erreicht
01391 69 02           adc #$02        Endadresse mit 2 addieren und
01393 8d 10 12        sta txttop      in BASIC-Textendezeiger ablegen
01396 a5 73           lda fbufpt+1    High-Byte mit eventuellem Übertrag
01398 69 00           adc #$00        addieren
0139a 8d 11 12        sta txttop+1    und in High-Byte des Zeigers speichern
0139d 4c fa 51        rts             fertig!

013a0   ===> eigene BASIC-Befehlswort-Tabelle mit $00 als <===
013a0                    Ende-Kennung
013a0
013a0 4f 4c c4  cmdlst .byte 'o1D'    BASIC-Befehlswort OLD
013a3 00               .byte eom

013b0 ===> eigene Adreßliste der erweiterten <===
013b0                   BASIC-Befehle
013b0
013b0 5f 13     adrlst .word oldcmd-1 Adresse der OLD-Routine minus 1, da die
                                      Adresse vom Stack gelesen wird
```

Die Initialisierung der »OLD«-Routine. Um den »OLD«-Befehl zu aktivieren, muß das obenstehende Programm in den Speicher geladen werden. Danach kann man es durch die BASIC-Befehle

```
BANK 0 : SYS DEC(''1300'')        »RETURN«
```

bzw. durch

```
BANK 0 : SYS 4864                 »RETURN«
```

initialisieren. Nach der Initialisierung kann man dann durch die Eingabe von

OLD »RETURN«

ein durch den »NEW«-Befehl gelöschtes BASIC-Programm reaktivieren. Es ist jedoch darauf zu achten, daß zwischen der Löschung des BASIC-Programmes und dem Reaktivieren keine Operationen geschehen, die ein BASIC-Programm ändern würden, wie etwa das Eingeben einer neuen BASIC-Programmzeile oder ähnliches.

1.9 Tips und Tricks

Unter dieser Rubrik sind einige Feinheiten des »BASIC 7.0« erläutert.

1.9.1 Der »FRE()«-Befehl

Der BASIC-Befehl »FRE()« wurde bereits in früheren BASIC-Versionen implementiert. Im »BASIC 7.0« kann dieser Befehl mit zwei möglichen Parametern zur Ausführung gebracht werden. »FRE(0)« gibt lediglich die Anzahl der Bytes aus, die in der RAM-0-Speicherseite für ein BASIC-Programm noch verfügbar sind. »FRE(1)« gibt die Anzahl der in der RAM-1-Speicherseite zur Verfügung stehenden Bytes als Ergebnis zurück. Bevor jedoch der freie Speicherplatz von »FRE(1)« berechnet wird, führt das »BASIC 7.0« die sogenannte »Garbage Collection« aus. »Garbage Collection« bedeutet Müllbeseitigung. Diese ist notwendig, da der »C 128« bei einer Manipulation der Stringvariablen-Inhalte diese nicht sofort in der entsprechenden Stringvariablen vornimmt, sondern lediglich den Zeiger auf den alten Inhalt löscht, den neuen Inhalt vor die Eintragung mit der niedrigsten Adresse schreibt und den vorher gelöschten Zeiger mit der Adresse der neuen Eintragung versieht. Damit bleibt der alte String im Speicher bestehen und der verfügbare Speicherplatz nimmt um die Länge des neu eingetragenen Strings ab. Selbst das Ersetzen nur eines Zeichens in einem String, bei gleichbleibender Stringlänge, führt zu dem eben Beschriebenen. Der so entstehende Stringmüll wird spätestens dann beseitigt, wenn der Speicherplatz für weitere hinzukommende Stringeinträge nicht mehr ausreichend ist. Durch den Befehl »FRE(1)« wird ebenfalls die »Garbage Collection« ausgelöst, so daß der danach berechnete freie Speicherplatz die real verfügbare Anzahl von Bytes wiedergibt.

FRE( )-Spezial. Die Autoren des »BASIC 7.0« haben im BASIC-ROM eine verschlüsselte Nachricht versteckt. Diese kann man mit Hilfe des »FRE(0)«-Befehls sichtbar machen. Jedoch ist es notwendig, vorher einige Zeiger zu *verbiegen*. Durch die Eingabe folgender Befehlszeile werden die entsprechenden Zeiger mit neuen Werten belegt. Wichtig zu wissen ist, daß die Befehle alle in einer Zeile eingegeben werden müssen und der 40-

Zeichen-Modus eingestellt ist. ACHTUNG! Nach dieser Aktion muß beim »C 128« ein Reset durchgeführt werden.

```
POKE4624,255:POKE4625,255:POKE4626,68:POKE4627,59:POKE53,44: ?FRE(0)
```

Nachdem die »RETURN«-Taste betätigt wurde, sollte diese Meldung am Bildschirm erscheinen:

```
Brought to you by...

Software
   Fred Bowen
   Terry Ryan
   Von ertwin

Hardware
   Bil Herb
   Dave Haymi
   Frank palaia

Link arms, don't make them
```

1.9.2 Der »RENUMBER«-Befehl

Im »BASIC 7.0« lassen sich die Zeilen eines BASIC-Programmes mit Hilfe des »RENUMBER«-Befehls umnumerieren. Dieser Befehl hat jedoch einen sogenannten *Bug*, einen Fehler. Normalerweise kann eine BASIC-Zeilennummer maximal den Wert 63999 haben. Folglich sollte auch der »RENUMBER«-Befehl eine BASIC-Programmzeile auf diesen Wert numerieren können. Die höchste Zeilennummer, welche der »RENUMBER«-Befehl jedoch produziert, ist 63743!

1.9.3 Der »MID$«-Befehl

Der »MID$«-Befehl ist ebenfalls aus Vorgängerversionen des »BASIC 7.0« bekannt. Neu ist jedoch, daß dieser Befehl auf der Zuweisungsseite eines Ausdrucks stehen kann und ohne Fehlermeldung verarbeitet wird. Es wird dadurch möglich, ein oder mehrere Zeichen in einem vorhandenen String auszutauschen bzw. einzufügen, ohne die üblichen Manipulationen durch String-Additionen mit den »LEFT$«- und »RIGHT$«-Befehlen durchführen zu müssen. Um beispielsweise den String »Hallo« in »Hello« durch den Austausch des Buchstabens »a« durchzuführen, kann man jetzt statt

```
A$=>>Hallo<<
A$=LEFT$(A$,1)+>>e<<+RIGHT$(A$,3)
```

folgendes schreiben:

```
A$=>>Hallo<<
MID$(A$,2,1)=>>e<<
```

1.9.4 Das Horner-Schema

Die interne Verarbeitung von Gleitkommawerten - speziell mit Funktionen wie »EXP()«, »SIN()«, »LOG()«, etc. - geschieht mittels Polynom-Arithmetik. Unter diesem Begriff versteht man die Näherungsrechnung mit Hilfe von Reihenentwicklungen mathematischer Funktionen. Diese Reihen bzw. Polynome haben folgende Form:

F(x) = a0 + a1*x + a2*x!2 + a3*x!3 + ... + aN*x!N

Die Ausrufungszeichen (»!«) in dem oben gezeigten Polynom stehen als Symbol für die Potenzierung von Zahlen!

a0 - aN sind reelle Koeffizienten, N ist eine natürliche Zahl. Die Koeffizienten a0 - aN und die Variable x (keine BASIC-Variable) werden in dem oben besprochenen Gleitkommaformat (5 Byte) verarbeitet. Das sogenannte Horner-Schema beruht darauf, die Berechnung von Polynomen n-ten Grades (n bezieht sich auf die höchste im Polynom vorkommende Potenz) zu erleichtern und zu beschleunigen. Die oben stehende Funktion sieht nach einer mathematischen Umformung in das Horner-Schema wie folgt aus:

F(x) = a0 + x*(a1 + x*(a2 + x*(a3 + ... + x*(aN-1 + aN*x) ...)))

Das bedeutet, wenn man die Potenzierung einer Zahl als mehrfache Multiplikation mit der gleichen Zahl betrachtet, daß bereits bei einem Polynom 4-ten Grades nach dem Horner-Schema lediglich »4« statt wie mit der oben stehenden normalen Funktion »10« Multiplikationen nötig sind, um das Polynom zu berechnen.

Dieses Horner-Schema wird - wie erwähnt - speziell für mathematische Funktionen verwendet. Im BASIC-ROM sind Tabellen vorhanden, die Koeffizienten für verschiedene Polynome enthalten. Die soll hier am Beispiel der BASIC-Funktion »EXP()« gezeigt werden.

Im BASIC-ROM-Listing finden Sie ab der Adresse »$9005« eine Tabelle mit Gleitkommakonstanten. Diese werden als Koeffizienten für ein Polynom 7-ten Grades verwendet, das, mit einem Wert »x« berechnet, als Resultat einen Näherungswert für die Funktion »EXP(x)« liefert. Die Funktion »EXP()« steht für die natürliche Exponentialfunktion zur »Basis e«=»2,7182815...«. Die Berechnung der Zahl »e« in BASIC wird durch »EXP(1)« erreicht. Addiert man die gegebenen Koeffizienten und multipliziert das Ergebnis mit der Zahl 1/log(2), so erhält man einen groben Näherungswert der Zahl »e«. Nach dieser Methode arbeitet auch das »BASIC

7.0«. Jedoch werden intern noch Fehlerkorrekturen vorgenommen. Die Zahl 1/log(2) ist ebenfalls als Gleitkommazahl im ROM gespeichert.

BASIC-Programm zum Horner-Schema. Will man in BASIC mit Polynomen arbeiten, empfiehlt sich auch hier das Horner-Schema, da damit auch im BASIC-Programm weniger komplizierte Berechnungen, speziell im Zusammenhang mit Potenzierungen, auftreten. Für ein BASIC-Programm bedeutet dies eine erhebliche Geschwindigkeitssteigerung.

Geben Sie einfach das folgende BASIC-Programm in Ihren »C 128«, und starten Sie es mit »RUN«. Sie werden überrascht sein, wie groß der Geschwindigkeitsvorteil ist. Die Ausrufungszeichen »!« in Zeile 130 stehen für das »Pfeil nach oben«-Zeichen, das im »BASIC 7.0« für die Potenzierung von Zahlen verwendet wird!

```
100 REM =====> Standardprogrammierung <======
110 TI$ = >>000000<<
120 FOR X = 1 TO 100
130     Y = 6 + 5*X + 4*X!2 + 3*X!3 + 2*X!4
140 NEXT
150 PRINT TI

160 REM ======> Horner-Schema <======
170 TI$ = >>000000<<
180 FOR X = 1 TO 100
190     Y = 6 + X*(5 + X*(4 + X*(3 + 2*X)))
200 NEXT
210 PRINT TI

FAST              »RETURN«
RUN               »RETURN«
 534
 64
```

Das Ergebnis zeigt, daß der Programmteil, der mit dem Horner-Schema programmiert wurde, weniger als eine Sekunde braucht, während die »normale« Funktion über 5 Sekunden benötigt.

2 GRUNDSÄTZLICHES ZUM ASSEMBLER UND DEN ASSEMBLERLISTINGS

Die folgenden Assemblerlistings wurden mit dem Hard- und Softassembler-/Reassemblersystem erstellt. Sie enthalten, ausführlich dokumentiert, das gesamte BASIC des »C 128« im 128er Modus. Zum besseren Verständnis über die Arbeitsweise und Leseart sind hier nun einige Informationen gegeben.

Die folgenden Listings sind reine Assemblerlistings und nicht etwa kommentierte Disassembler-Ausdrucke. Das Listing liegt also in ASCII-Sourcefiles vor und wurde mit einem professionellen Assembler assembliert und ausgedruckt. Dabei entsteht ein komplett lauffähiges Programm, das bis aufs Bit dem Original-Betriebssystem entspricht.

Die verwendeten Labelnamen sind - soweit bekannt - die Original-Labelnamen, wie sie auch von Commodore verwendet wurden. Die restlichen Namen wurden so gewählt, daß möglichst aufgrund des Namens bereits auf die mögliche Funktion der Routine geschlossen werden kann.

Für alle, die bisher noch keine Erfahrungen mit einem Assembler haben, sondern ihre Programme zum Beispiel mit Hilfe eines Monitorprogrammes geschrieben haben, hier einige Hinweise zur Arbeitsweise eines Assemblers und zu Eigenheiten des verwendeten Assemblersystems.

2.1 Einführung in die Assemblerprogrammierung

Den Vorgang, ein mnemonisches bzw. symbolisches Computerprogramm in einen prozessortypischen Maschinen-Code umzusetzen, nennt man Assemblierung. Ein Programm, welches die Assemblierung durchführt, nennt man Assembler. Die dabei verwendeten Symbole, Befehle und Anwendungsregeln sind die Assemblersprache. Generell wird jeder Begriff einer Assemblersprache vom Assembler in eine Maschineninstruktion übersetzt. Hier ist der Unterschied zum Compiler, der aus einem einzigen Begriff

eine Anzahl von Maschineninstruktionen erzeugt. Wird ein Assembler auf einem anderen Computer (anderen Mikroprozessor) betrieben, als auf dem, für den der Maschinen-Code generiert werden soll, dann spricht man von einem Cross-Assembler.

Im Prinzip verwenden Computer, und damit auch alle Mikroprozessoren, das duale Zahlensystem (Basis 2) und binäre Codierung für Daten und Befehle. Für den Menschen als Anwender und speziell als Programmierer eignen sich zur Darstellung andere Zahlensysteme besser, wie das oktale (Basis 8), dezimale (Basis 10) oder das hexadezimale (Basis 16) System. Für die Maschinen- bzw. Assemblerprogrammierung scheint die hexadezimale Darstellung am geeignetsten.

Die verschiedenen Zahlensysteme sollen an dem folgenden Befehlsbeispiel gezeigt werden:

Lade den Akkumulator direkt mit dem Wert »Hex 56« - Befehls-Code »Hex A9«

Zahlendarstellung	Befehl	Wert
Hexadezimal	A9	56
Dezimal	169	86
Dual	10101001	01010110

Die numerische Darstellung wie im Beispiel ist in der Anwendung zu mühselig, da der Programmierer dann alle Befehle und deren numerischen Wert wissen müßte. Es wurden deshalb symbolische Ausdrücke, sogenannte Mnemonics, entwickelt.

Bei Verwendung dieser Mnemonics kann das vorhergehende Beispiel folgendermaßen dargestellt werden:

Zahlendarstellung	Befehl und Wert
Hexadezimal	LDA #56
Dezimal	LDA #86
Dual	LDA #%01010110

Bei dieser Darstellung ist LDA der Mnemonic für den Befehl Lade Akku = $A9, das »#« steht für die Adressierungsart »direkt« und die Bezeichnungen $ für hexadezimal und % für dual. Eine dezimale Angabe erfolgt ohne Vorzeichen.

Der Assembler hat nun die Aufgabe, das Mnemonic »LDA #« in die numerische Form »$A9« zu übersetzen.

Das mnemonische Programm wird als Source-Code oder Quell-Code, das vom Assembler in Maschinen-Code umgesetzte Programm als Objekt-Code

bezeichnet. Der Prozessor kann nur den Objekt-Code verstehen und ausführen.

So wie im Beispiel gezeigt, hat jeder Maschinen-Code einen symbolischen Namen, den Operations-Code, kurz mit Op-Code bezeichnet. Wie bereits ausgeführt, verfügt der Mikroprozessor »650x« und damit auch der »8502« über 56 verschiedene Op-Codes. Bei vielen dieser Op-Codes sind zusätzlich mehrere Adressierungsarten möglich.

Die meisten 65xx-Befehle (Op-Codes) benötigen als zusätzliche Angabe Daten, mit denen der Befehl auszuführen ist. Diese Daten werden als Operand bezeichnet. Der Operand kann auch, frei wählbar, symbolisch angegeben sein. Man kann als Operand auch einen ASCII-Code angeben. Zur Kennzeichnung des ASCII-Codes muß dieser zwischen zwei Apostrophe gestellt werden (z.B. 'a').

Symbolische Adressen oder Werte werden als Label bezeichnet. Um eventuelle Ansprünge aus dem Programm zu einem Befehl kenntlich zu machen, kann auch vor dem Op-Code linksbündig ein Label stehen.

Beispiel:

```
Label     Op-Code     Operand

anfang    LDA         $90
          STA         abc
          LDA         'a'
...       ...         ...
ende      STA         zero90
```

2.2 Erklärung des Assemblerlistings

Das Assemblerlisting hat folgenden grundsätzlichen Aufbau:

Zeilennummer Hex-Adresse Objekt-Code Quell-Code Kommentar

2.2.1 Zeilennummernfeld

Die Zeilennummer ist fünfstellig. Nach jeder Assemblerzeile wird die Zeilennummer um 1 erhöht. Sie ist im Zusammenhang mit der später beschriebenen Cross-Referenzliste (Verweistabelle) sehr wichtig.

2.2.2 Hex-Adressenfeld

An der nächsten Position in der Assemblerzeile steht die hexadezimale Adresse. Da der Prozessor nur 64 KByte direkt adressieren kann, genügt hier eine vierstellige Darstellung von $0000 bis $FFFF. Je nachdem, ob in

der Zeile ein 1-Byte-, 2-Byte- oder 3-Byte-Befehl steht, wird die Adresse in der Folgezeile um 1, 2 oder 3 erhöht (inkrementiert).

2.2.3 Objekt-Code-Feld

Hier steht der vom Assembler erzeugte Maschinen-Code, der vom Prozessor direkt verarbeitet werden kann. Es ist hier Platz für drei Bytes reserviert. Stehen in der Zeile Deklarationen, dann ist dieser Platz frei, weil kein Maschinen-Code erzeugt wird.

2.2.4 Quell-Code-Feld

Der Quell-Code kann in verschiedener Form im Assemblerlisting erscheinen.

Deklaration und Zuweisung von Label. Wird im Assemblerlisting einem Label ein bestimmter Wert zugewiesen, dann erfolgt dies, ähnlich wie in BASIC, mit dem Gleichheitszeichen »=«.

Beispiel: test=$85

In diesem Beispiel wird dem Label mit dem Namen »test« der hexadezimale Wert »85« zugewiesen. Wird nun im Listing z.B. das Label »test« als Operand verwendet, dann wird im Objekt-Code hierfür der Wert »85« eingesetzt.

Eine weitere Möglichkeit besteht mit »* = Adresse«. Man kann hiermit problemlos ab diesem Zeitpunkt eine neue Programmadresse vorgeben.

Beispiel: *=$1234

Ab diesem Zeitpunkt wird die Programmadresse auf hexadezimal »$1234« gesetzt.

Beispiel: *=*+$100

War bei diesem Beispiel die Adresse z.B. »$F100«, so wird nach dieser Zeile die Adresse auf »F200« gesetzt.

Linksbündiges Label. Beginnt der Quell-Code mit einem Label (linksbündiges Label) und darauffolgendem Mnenomic, so wird diesem Label vom Assembler die in dieser Zeile aktuelle Adresse zugewiesen.

Mnemonics. Die Mnemonics werden aus Gründen der Übersichtlichkeit an einer eigenen Tabulatorposition im Listing ausgegeben. Es ist dadurch sehr leicht eine Unterscheidung vom linksbündigen Label möglich.

Assemblerdirektiven. Die Assemblerdirektiven veranlassen den Assembler, bestimmte Funktionen auszuführen. Die Direktiven stehen stets im Feld »Quell-Code« und beginnen mit einem Punkt.

.byte label	Der Assembler setzt im Objekt-Code den Wert des Labels (1 Byte) ein.
.byte nn,nn	Der Assembler setzt im Objekt-Code die Werte der hinter der Direktive folgenden Bytes ein.
.byte 'abcd'	Der Assembler setzt im Objekt-Code die Werte der hinter der Direktive folgenden Bytes ein.
.byte 'abcd'	Der Assembler setzt im Objekt-Code die ASCII-Codes der Zeichen 'abcd' ein (4 Bytes)
.byte "abcd"	Der Assembler setzt im Objekt-Code die Bildschirm-Codes der Zeichen 'abcd' ein.
.byte <label	Der Assembler setzt im Objekt-Code den niederwertigen Teil der Adresse, = Low-Byte des Adreß-Labels (1 Byte), ein.
.byte <label	Der Assembler setzt im Objekt-Code den höherwertigen Teil der Adresse, = High-Byte des Adreß-Labels (1 Byte), ein.
.word label	Der Assembler setzt im Objekt-Code den Wert des Adreß-Labels (2 Byte) in der Reihenfolge Low-Byte, High-Byte ein.

Operand. Der Operand kann als Label, aber auch als Wert angegeben sein. Werte ohne Vorzeichen sind Dezimalwerte, Werte mit vorgestelltem »$« sind Hexwerte und Werte mit »%« sind Binärwerte (achtstellig).

Der Operand kann auch berechnet werden, wie beispielsweise »test + $80«. Es wird hierbei generell ohne Beachtung von Rechenregeln von links nach rechts gerechnet.

Als Operand können auch in Apostrophe eingeschlossene ASCII-Zeichen eingesetzt sein beziehungsweise Zeichen im Commodore-Bildschirmformat, eingeschlossen in Anführungszeichen. Bei Immediate-Befehlen (der Operand beginnt mit einem »#«) kann auch durch ein folgendes »<« das Low-Byte bzw. durch ein »>« das High-Byte einer Adresse im Objekt-Code eingesetzt werden.

2.2.5 Kommentarfeld

Im Kommentarfeld stehen Hinweise und Erklärungen, die sich auf diese Zeile beziehen. Reicht der Platz für die Kommentare nicht aus, dann wird in der nächsten Zeile im Kommentarfeld fortgesetzt. Die Zeilennummer erhöht sich in solchen Fällen nicht zusätzlich.

2.3 Cross-Referenzliste beziehungsweise Verweistabelle

Anschließend an das Assemblerlisting folgt eine Cross-Referenzliste, eingedeutscht auch Verweistabelle genannt. Sie enthält in alphabetischer Reihenfolge alle im Assemblerlisting vorkommenden Label-Namen (symbolische Zuordnung einer Adresse oder Wert), die Adresse als Hex-Zahl und darauf folgend, durch einen Doppelpunkt getrennt, die fünfstelligen Zeilennummern, in denen das Label vorkommt.

Mit Hilfe der Cross-Referenzliste ist es beispielsweise jederzeit möglich festzustellen, ob und von wo eine bestimmte Unterroutine aus anderen Programmteilen angesprungen wird, bzw. wo eine bestimmte Speicherstelle definiert, benutzt oder verändert wird.

Eine zusätzliche Information bieten die Zeichen »#« und »$«. Folgt direkt hinter einer Zeilennummer ein »#«, dann wird in der angegebenen Zeile das Label definiert. Zur Erklärung ein Beispiel einer Assemblerzeile, in der ein Label definiert wird.

```
Zeile  Adr.  Obj.-Code  Quell-Code

00123  0000             flag = $CA      Ein-/Ausgabe-Flag
```

Es wird hier dem Label »flag« der hexadezimale Wert »CA« zugewiesen.

Eine weitere Möglichkeit einer Label-Definition ist ein sogenanntes linksbündiges Label. Hier wird dem Label die gerade aktuelle Programmadresse als Wert zugewiesen. Auch hierzu ein Beispiel einer möglichen Assemblerzeile:

```
Zeile  Adr.  Obj.-Code  Quell-Code

06065  F011  8A         beginn TXA      Inhalt von X-Reg. in Accu
```

In diesem Beispiel wird dem Label »beginn« der hexadezimale Wert »F011« der aktuellen Adresse zugewiesen. Wird nun irgendwo im Listing dieses Label verwendet, dann setzt der Assembler dafür den Wert »$F011« ein.

Folgt in der Cross-Referenzliste direkt hinter der Zeilennummer ein »$«, dann wird in dieser Zeile im Assemblerlisting der Inhalt der Speicherstelle, die durch das Label definiert ist, durch einen entsprechenden Prozessorbefehl geändert, zum Beispiel:

```
Zeile  Adr.  Obj.-Code  Source-Code

06156  F169  85 55      STA zwert        Zwischenwert für Zähler
```

Es wird hier in die Speicherstelle mit der Adresse »$55« und dem Label-Namen »zwert« der Inhalt das Akkumulators eingeschrieben.

Die folgenden Zeilen zeigen im Beispiel den Aufbau der Cross-Referenzliste bzw. Verweistabelle:

= Definition, $ = Schreibzugriff, leer = Lesezugriff/Sprung

```
Label   Adr.    Zeilennummern

beginn = $F011: 00955,00971,06065#, 06269
:
:
flag   = $CA  : 00123#, 00845, 0139$, 0180$, 04960
:
:
zwert  = $55  : 00089#, 02790, 03531$, 03840, 06156$
```

Handhabungsbeispiel. Beim Durcharbeiten des Assemblerlistings ist es für Sie von großem Wert zu wissen, ob z.B. die Unterroutine »beginn« (linksbündiges Label) mehrfach verwendet wird. Dazu schlagen Sie in der nach Label-Namen sortierten Cross-Referenzliste unter »beginn« nach. Dort finden Sie, wie unser Beispiel zeigt, den zugewiesenen Wert des Labels, hexadezimal »$F011«, weiterhin die Zeilennummern und außerdem, durch ein »#« markiert, die Zeilennummer, in der das Label definiert wurde. In unserem Beispiel Zeile 06065.

3 C-128-ROM-LISTING: BASIC-BETRIEBSSYSTEM

Pass 1

```
00008  0000                .lib basic.lab
00764  0000                .lib basic.cpu
00820  0000                .lib basic.vic
00959  0000                .lib basic.sid
01092  0000                .lib basic.mmu
01250  0000                .lib basic.vdc
01416  0000                .lib basic.cia
01609  0000                .lib basic.dma
01631  0000                .lib basic.init
02192  430a                .lib basic.crunch
02466  4417                .lib basic.reslst
02714  46fc                .lib basic.dispatch
02918  4846                .lib basic.errlst
03093  4a9f                .lib basic.exec
03809  4daf                .lib basic.main
04533  50e2                .lib basic.cmd_1
05286  53c6                .lib basic.varia
05617  553a                .lib basic.cmd_2
06269  57f4                .lib basic.cmd_3
06989  5af0                .lib basic.renum
07717  5df9                .lib basic.cmd_4
08583  61a8                .lib basic.graph_1
09288  658d                .lib basic.graph_2
10227  69e2                .lib basic.graph_3
10868  6c4f                .lib basic.sprite_1
11264  6de1                .lib basic.sound_1
11926  7046                .lib basic.sound_2
12695  7335                .lib basic.boot
12763  7372                .lib basic.sprite_2
13748  77d7                .lib basic.eval
14951  7c66                .lib basic.array
15489  8000                .lib basic.functs
16654  8434                .lib basic.math_1
17082  85ae                .lib basic.string
17641  87f1                .lib basic.math_2
18360  8aec                .lib basic.math_3
18919  8d22                .lib basic.math_4
19749  90d0                .lib basic.io
20162  9299                .lib basic.garbcol
```

```
20525  9409                    .lib basic.math_5
20771  9520                    .lib basic.using
21568  99c1                    .lib basic.instr
21727  9a74                    .lib basic.graph_4
22327  9d24                    .lib basic.graph_5
23148  a07e                    .lib basic.diskb_1
23945  a3f8                    .lib basic.diskb_2
24988  a84d                    .lib basic.irq
25501  af00                    .lib basic.jmptbl
```

Pass 2

```
00001  0000  put "@basic.lib"
00002  0000   Version 24.05.86  -omb-
00003  0000
00004  0000
00005  0000  ===> library-file <===
00006  0000
00007  0000
00008  0000                    .lib basic.lab
00009  0000  put"@basic.lab"
```

```
00011  0000
00012  0000  ===> Konstanten, Codes <===
00013  0000
00014  0000              _eom   = 0              end-of-message
00015  0000              _blink = 2              blink on
00016  0000              _stpky = $03            Code Stopp-Taste
00017  0000              _lf    = $0a            line feed
00018  0000
00019  0000              _cr    = $0d            carriage return
00020  0000              _sftcr = $8d            shift carriage return
00021  0000
00022  0000              _uplo  = $0e            Code Umsch. Groß/Kleinschrift
00023  0000              _upgrp = $8e            Code Umsch. Groß/Graphik
00024  0000
00025  0000              _crsdn = $11            cursor down
00026  0000              _crsup = $91            cursor up
00027  0000
00028  0000              _rvson = $12            reverse on
00029  0000              _rvsof = $92            reverse off
00030  0000
00031  0000              _home  = $13            cursor home
00032  0000              _clr   = $93            screen clear
00033  0000
00034  0000              _esc   = $1b            escape
00035  0000
00036  0000              _crite = $1d            cursor rechts
00037  0000              _cleft = $9d            cursor links
00038  0000
00039  0000              _spc   = $20            Space
00040  0000
00041  0000              _pi    = $ff            Code für 'pi'
00042  0000
00043  0000              _not   = 256
00044  0000
00045  0000
00046  0000              _deflt = 10             Ersatzwert für RENUMBER
00047  0000
00048  0000              _skip1 = $24            Opcode zum Überl. des folg. Bytes
00049  0000              _skip2 = $2c            Opcode zum Überl. der folg. zwei
                                                 Bytes
00050  0000
00051  0000              _tkofs = $80            Offset für BASIC-Token
00052  0000
00053  0000              _lnkdy = $0101          Dummy Linkadresse
00054  0000
00055  0000              _beats = $0480          Wert für 1/1 Note
00056  0000              _hedln = 4              Länge PGM-Zeilen Kopf
00057  0000              _tidln = 6              Länge TI$-String
00058  0000
00059  0000              _tmpst = $1b            Zeiger in String-Stapel
00060  0000
00061  0000              _expcn = 129            Offset für Exponent FAC
00062  0000
00063  0000              _graph = $2400          9 KB = Größe Graphik RAM
00064  0000
00065  0000
00066  0000
```

```
00066  0000  ===> Definition Zeropagelabels <===
00067  0000
00068  0000            dummy  = $00           veränd. Ersatz-ZP-Adresse
00069  0000            srchtk = $02           Speicher für Such Token
00070  0000
00071  0000            bnk    = $02           Bank Nummer
00072  0000
00073  0000            _pgmbk = 0             PGM RAM
00074  0000            _varbk = 1             VAR RAM
00075  0000            _sysbk = 15            SYS ROM
00076  0000
00077  0000            pchi   = $03           Puffer 8502 CPU ADDRH PC
00078  0000            pclo   = $04           Puffer 8502 CPU ADDRL PC
00079  0000            sreg   = $05           Puffer 8502 CPU SR
00080  0000            areg   = $06           Puffer 8502 CPU AC
00081  0000            xreg   = $07           Puffer 8502 CPU XR
00082  0000            yreg   = $08           Puffer 8502 CPU YR
00083  0000
00084  0000            charac = $09           Suchzeichen
00085  0000            integr = $09
00086  0000
00087  0000            endchr = $0a           Abbruchzeichen
00088  0000            numflg = $0a           Flag für Zeilennr. gelesen
00089  0000
00090  0000   = 0  nicht gelesen
00091  0000   <> 0  gelesen
00092  0000
00093  0000            trmpos = $0b           Tabulator-Spalte
00094  0000            bverck = $0c           Flag LOAD=0, VERIFY=1 (BASIC)
00095  0000
00096  0000            count  = $0d
00097  0000            fconc  = $0d
00098  0000            movlpt = $0d           Verschiebezeiger Lesen RENUMBER
00099  0000
00100  0000            dimflg = $0e           Flag DIM/Zeiger suchen
00101  0000
00102  0000   <> 0  Feld dim. (1. Zeich. VAR NAM)
00103  0000    = 0  Zeiger auf Var. suchen
00104  0000
00105  0000            valtyp = $0f           Flag Variablentyp
00106  0000
00107  0000            _numer = 0             Flag numerischer Wert
00108  0000            _strng = 255           Flag String
00109  0000
00110  0000            intflg = $10           Flag für Gleitkomma/Integer
00111  0000
00112  0000   7.......  1 = INT VAR, 0 = GK VAR
00113  0000
00114  0000            dores  = $11           Flag für DATA, LIST, garbage
                                              collection
00115  0000            sublfg = $12           Flag Zulassen/Sperren INT-Var.
00116  0000
00117  0000   7.......  1=INT Var. gesperrt, 0=zugelassen
00118  0000
00119  0000            inpflg = $13           Flag Herkunft Aufruf inloop
00120  0000
00121  0000            _input = $00
00122  0000            _get   = $40
```

```
00123  0000            _read  = $98
00124  0000
00125  0000            tansgn = $14          Flag für TAN-Vorzeichen
00126  0000            channl = $15          aktiver I/O-Kanal
00127  0000
00128  0000            _keybd = 0            Kanalnummer Tastatur
00129  0000            _cass  = 1            Kanalnummer Cassette
00130  0000            _rs    = 2            Kanalnummer RS232
00131  0000            _scrn  = 3            Kanalnummer Bildschirm
00132  0000
00133  0000            linnum = $16
00134  0000            temppt = $18          Zeiger in den String-Stapel
00135  0000            lastpt = $19          ADDRL letzt benutzte String-Adresse
00136  0000            tempst = $1b          Descriptor-Stapel
00137  0000            index1 = $24
00138  0000            index2 = $26
00139  0000
00140  0000            resho  = $28          RES: M4
00141  0000            resmoh = $29          RES: M3
00142  0000            resmo  = $2a          RES: M2
00143  0000            reslo  = $2b          RES: M1
00144  0000
00145  0000            txttab = $2d          ADDRL Untergrenze akt. PGM
00146  0000            vartab = $2f          ADDRL Zeiger Anfang einf. Variable
00147  0000            arytab = $31          ADDRL Zeiger Anfang Feld-Tabelle
00148  0000            strend = $33          ADDRL+1 Obergr. ARRAY Tabelle
00149  0000            fretop = $35          ADDRL Zeiger Anfang String-Variable
00150  0000            frespc = $37          ADDRL Zeiger auf neuen String
00151  0000            mxmem1 = $39          ADDRL Obergrenze VAR RAM
00152  0000            curlin = $3b          Low-Byte akt. Zeilennummer
00153  0000            txtptr = $3d          ADDRL chrget-Zeiger
00154  0000            fndpnt = $3f          ADDRL BASIC-Stapel Suchzeiger
00155  0000            datlin = $41          ADDRL DATA-Zeilennummer
00156  0000            datptr = $43          ADDRL Zeiger auf DATA (READ)
00157  0000            inpptr = $45          ADDRL Zeiger auf Input-Quelle
00158  0000            varnam = $47          erstes Zeichen akt. Variablenname
00159  0000            varpnt = $49          ADDRL Zeiger auf Variable im RAM
00160  0000            lstpnt = $4b
00161  0000
00162  0000            vartxt = $4d          ADDRL Zeiger VAR in PGM
00163  0000            opptr  = $4d          Zeiger auf Prio. Flag
00164  0000
00165  0000            opmask = $4f          vom akt. Operator erzeugte Bit-Maske
00166  0000
00167  0000            defpnt = $50          ADDRL Zeiger FN Variable
00168  0000            grbpnt = $50          ind. Zeiger für garbage collect
00169  0000            tempf3 = $50
00170  0000
00171  0000            dscpnt = $52          ADDRL Zeiger String-Descr.
00172  0000
00173  0000            _dscln = 7            Länge VAR Descriptor
00174  0000            _aryhd = 5            Länge ARY Kopf
00175  0000            _aryln = 3            Länge ARY Desc.
00176  0000
00177  0000            helper = $55          Flag HELP / LIST
00178  0000            jmper  = $56          jmp-Opcode (Funktions-Routinen)
00179  0000            oldov  = $58
00180  0000
```

```
00181  0000            ptarg1 = $59            Zeiger Descr.Durchsuch String INSTR
00182  0000            tempf1 = $59            TMP FAC1
00183  0000            tempf2 = $5e            TMP FAC2
00184  0000
00185  0000            highds = $5a
00186  0000            arypnt = $5a
00187  0000
00188  0000            ptarg2 = $5b            Zeiger Descr. Suchstring INSTR
00189  0000
00190  0000            hightr = $5c
00191  0000            rostln = $5c            Low-Byte alte Startzeile RENUMBER
00192  0000
00193  0000
00194  0000            str1   = $5d            temp. Stringdescr.
00195  0000            str2   = $60            temp. Stringdescr.
00196  0000
00197  0000            tenexp = $60
00198  0000
00199  0000            dptflg = $61
00200  0000            grbtop = $61
00201  0000            expsgn = $62
00202  0000
00203  0000            deccnt = $5f
00204  0000            str2   = $60            temp. Stringdescr.
00205  0000            lowtr  = $61
00206  0000
00207  0000            positn = $63            Zeiger in Durchsuch String
00208  0000            dsctmp = $63            tmp. Stringdescriptor
00209  0000
00210  0000            match  = $64
00211  0000            indice = $66
00212  0000
00213  0000            facexp = $63            FAC: EX
00214  0000            _ex    = facexp-facexp
00215  0000            facho  = $64            FAC: M4
00216  0000            _m4    = facho-facexp
00217  0000            facmoh = $65            FAC: M3
00218  0000            _m3    = facmoh-facexp
00219  0000            facmo  = $66            FAC: M2
00220  0000            _m2    = facmo-facexp
00221  0000            faclo  = $67            FAC: M1
00222  0000            _m1    = faclo-facexp
00223  0000            facsgn = $68            FAC: SG
00224  0000            _sg    = facsgn-facexp
00225  0000            sgnflg = $69
00226  0000
00227  0000            argexp = $6a            ARG: EX
00228  0000            argho  = $6b            ARG: M4
00229  0000            argmoh = $6c            ARG: M3
00230  0000            argmo  = $6d            ARG: M2
00231  0000            arglo  = $6e            ARG: M1
00232  0000            argsgn = $6f            ARG: SG
00233  0000            arisgn = $70            Vorzeichen Ergebnis Operation
00234  0000            facov  = $71            Rundungsbyte
00235  0000
00236  0000
00237  0000            movspt = $6d            Verschiebezeiger Schreiben RENUMBER
00238  0000
```

```
00239  0000            strng1 = $70
00240  0000            strng2 = $72
00241  0000
00242  0000            fbufpt = $72            ADDRL Zeiger in Puffer
00243  0000            polypt = $72
00244  0000
00245  0000            autinc = $74            Schrittweite low für AUTO
00246  0000            mvdflg = $76            Flag GRAPHIC-RAM
00247  0000
00248  0000  $ff = GRAPHIC-RAM reserviert
00249  0000  $00 = nicht reserviert
00250  0000
00251  0000            mltus1 = $77
00252  0000            tmptk  = $77            temp. Token bei GET
00253  0000            midln  = $77            Länge Ersatzstring bei MID$
00254  0000            rnumfg = $77            Flag RENUMBER
00255  0000
00256  0000   0 = Zeilennr. ersetzen
00257  0000   1 = Test genügend RAM für neue Zeilennr.
00258  0000
00259  0000            hulp   = $78            PRINT USING : Zähler
00260  0000            midpt  = $78            Startspalte Ersatzstring bei MID$
00261  0000
00262  0000            syntmp = $79
00263  0000            lentmp = $79
00264  0000
00265  0000            dsdesc = $7a            Desc. DS$ (Länge)
00266  0000
00267  0000            _dsdln = 40             Länge DS$-String
00268  0000
00269  0000            tos    = $7d            ADDRL BASIC-Stapel Zeiger
00270  0000            runmod = $7f            Modus: $80=PGM, $40=OVL, $00=DIR
00271  0000            parsts = $80            DOS Syntax Byte 1
00272  0000
00273  0000            _fn1   = %00000001      1. Filename
00274  0000            _fn2   = %00000010      2. Filename
00275  0000            _la    = %00000100      log. Adresse
00276  0000            _fa    = %00001000      Primäradresse
00277  0000            _d1    = %00010000      1. Laufwerk
00278  0000            _d2    = %00100000      2. Laufwerk
00279  0000            _rcl   = %01000000      Record Länge
00280  0000            _rpl   = %10000000      replace @
00281  0000
00282  0000            parstx = $81            DOS Syntax Byte 2
00283  0000
00284  0000            _bnk   = %00000001      Bank
00285  0000            _stadr = %00000010      Startadresse
00286  0000            _endad = %00000100      Endadresse
00287  0000
00288  0000            oldstk = $82            Stapelzeiger für trap
00289  0000            colsel = $83            Farbquelle (0-3)
00290  0000            mulco1 = $84            Multi-Color Farbe 1
00291  0000            mulco2 = $85            Zusatzfarbe 2
00292  0000            forgnd = $86            Farbcode Vordergrund
00293  0000            scalex = $87            SCALE Low-Byte x-Wert (Abszisse)
00294  0000            scaley = $89            SCALE Low-Byte y-Wert (Ordinate)
00295  0000            stopnb = $8b            Flag für paint, Rahmenflag
00296  0000            grapnt = $8c            ADDRL Zeiger in Graphik-RAM
```

```
00297  0000            vtemp1 = $8e
00298  0000            vtemp2 = $8f
00299  0000
00300  0000            status = $90            I/O-Status ST
00301  0000
00302  0000    7.......  1 = Dev.not pres. oder Band-Ende
00303  0000    .6......  1 = End of File
00304  0000    ..5.....  1 = Prüfsummenfehler
00305  0000    ...4....  1 = Fatal Error
00306  0000    ....3...  1 = Block zu lang (Kass.)
00307  0000    .....2..  1 = Block zu kurz (Kass.)
00308  0000    ......1.  1 = Timeout Lesen
00309  0000    .......0  1 = Timeout Schreiben
00310  0000    00000000  Status ok
00311  0000
00312  0000            verck  = $93            Flag für LOAD/VERIFY
00313  0000    00000000  load
00314  0000    00000001  verify
00315  0000
00316  0000            ptr2   = $9f
00317  0000            sal    = $ac            ADDRL für LOAD/SAVE
00318  0000            sah    = $ad            ADDRH für LOAD/SAVE
00319  0000            fnlen  = $b7            Länge akt. Filename
00320  0000            sa     = $b9            akt. Sekundäradresse
00321  0000            fa     = $ba            akt. Primäradresse
00322  0000            fnadr  = $bb            ADDRL akt. Filename
00323  0000            ba     = $c6            akt. Bank für Speicher-Operationen
00324  0000            fnbank = $c7            Bank des akt. Filenamens
00325  0000            ndx    = $d0            Bytes im Tast.puffer
00326  0000
00327  0000            mode   = $d7            BS-Modus 40/80-Zeichen
00328  0000
00329  0000    00000000  40-Zeichen BS
00330  0000    10000000  80-Zeichen BS
00331  0000
00332  0000            graphm = $d8            Flag Text/Graphik/Split-BS
00333  0000
00334  0000    00000000  Text                                  GRAPH 0
00335  0000    00100000  HI-RES                                GRAPH 1
00336  0000    01100000  geteilter BS HI-RES/Text              GRAPH 2
00337  0000    10100000  HI-RES Multi-Color                    GRAPH 3
00338  0000    11100000  geteilter BS HI-RES Multi-Color/Text  GRAPH 4
00339  0000
00340  0000            scbot  = $e4            BS-Fenster: unterste Zeile
00341  0000            sctop  = $e5            BS-Fenster: oberste Zeile
00342  0000            sclf   = $e6            BS-Fenster: Spalte linker Rand
00343  0000            scrt   = $e7            BS-Fenster: Spalte rechter Rand
00344  0000
00345  0000            _rows  = 25             Vorgabe BS Zeilen
00346  0000            _col40 = 40             Vorgabe Spalten 40 Zeichen BS
00347  0000            _col80 = 80             Vorgabe Spalten 80 Zeichen BS
00348  0000
00349  0000            attrib = $f1            akt. Attribut
00350  0000
00351  0000    7.......  1 = Groß/Graphik, 0 = Klein/Groß
00352  0000    .6......  1 = RVS ein, 0 = aus (nicht benutzt)
00353  0000    ..5.....  1 = Unterstreichen ein, 0 = aus
00354  0000    ...4....  1 = Blinken ein, 0 = aus
```

```
00355  0000    ....3...  1 = Rot
00356  0000    .....2..  1 = Grün
00357  0000    ......1.  1 = Blau
00358  0000    .......0  1 = Intensität
00359  0000
00360  0000              insflg = $f6            AUTO-INSERT-Flag
00361  0000
00362  0000    11111111  AUTO-INSERT Modus ein
00363  0000    00000000  AUTO-INSERT Modus aus
00364  0000
00365  0000              lofbuf = $ff
00366  0000
00367  0000
00368  0000  ===> Definition RAM-Labels <===
00369  0000
00370  0000              stack  = $0100          CPU Stack
00371  0000              fbuffr = $0100
00372  0000
00373  0000  ===> DISK BASIC <===
00374  0000
00375  0000              xcnt   = $0110          DOS Schleifen-Zähler
00376  0000
00377  0000              dosf1l = $0111          Länge Filename 1
00378  0000              _fnam1 = $f1            DOS Kürzel für 2. Filenamen
00379  0000
00380  0000              dosds1 = $0112          1. Laufwerk
00381  0000              _drv1  = $d1            DOS Kürzel für 1. Laufwerk
00382  0000
00383  0000              dosf2l = $0113          Länge 2. Filename
00384  0000              _fnam2 = $f2            DOS Kürzel für 2. Filenamen
00385  0000
00386  0000              dosds2 = $0114          2. Laufwerk
00387  0000              _drv2  = $d2            DOS Kürzel für 2. Laufwerk
00388  0000
00389  0000              dosf2a = $0115          ADDRL 2. Filename
00390  0000              dosofl = $0117          ADDRL Untergrenze BLOAD/BSAVE
00391  0000              dosofh = $0119          ADDRL Obergrenze BLOAD/BSAVE
00392  0000
00393  0000              dosla  = $011b          logische Adresse
00394  0000              _defla = 0              Vorgabe la Floppy
00395  0000
00396  0000              dosfa  = $011c          Primäradresse
00397  0000              _deffa = 8              Vorgabe fa Floppy
00398  0000
00399  0000              dossa  = $011d          Sekundäradresse
00400  0000              _defsa = 0              Vorgabe sa
00401  0000              _cmdsa = 15             sa Floppy Kommandokanal
00402  0000              _talk  = %01100000      Offset sa für Talk
00403  0000
00404  0000              _sa    = $c2            DOS Kürzel für Sekundäradresse
00405  0000
00406  0000              dosrcl = $011e          Record Länge
00407  0000              _recln = $e0            DOS Kürzel für Record Länge
00408  0000
00409  0000              _rlrh  = $e2            DOS Kürzel für Record Nummer
                                                (low/high)
00410  0000
00411  0000              dosbnk = $011f          Bank Nummer
```

```
00412  0000
00413  0000            dosdid = $0120          ID erstes Zeichen
00414  0000            _id    = $d0            DOS Kürzel für ID
00415  0000
00416  0000            didchk = $0122          DOS ID Flag
00417  0000
00418  0000            _seqwr = $e1            DOS Kürzel für ',s,W'
00419  0000            _ovwr  = $f0            DOS Kürzel für @
00420  0000
00421  0000            plyflg = $0126          Flag PLAY Befehle
00422  0000
00423  0000  ===> PRINT USING <===
00424  0000
00425  0000            bnr    = $0123          PRINT USING : Zeiger auf Anfang
00426  0000            enr    = $0124          PRINT USING : Zeiger auf Ende
00427  0000            dolr   = $0125          PRINT USING : Dollar-Flag
00428  0000            flag   = $0126          PRINT USING : Komma-Flag
00429  0000            swe    = $0127          PRINT USING : Zähler
00430  0000            usgn   = $0128          PRINT USING : Vorzeichen Exponent
00431  0000            uexp   = $0129          PRINT USING : Zeiger auf Exponent
00432  0000            vn     = $012a          PRINT USING : Anzahl Stellen vor
                                               Dezimalpunkt
00433  0000            chsn   = $012b          PRINT USING : justify Flag
00434  0000            vf     = $012c          PRINT USING : # Position vor
                                               Dezimalpunkt
00435  0000            nf     = $012d          PRINT USING : # Position nach
                                               Dezimalpunkt
00436  0000            posp   = $012e          PRINT USING : +/- Flag
00437  0000            fesp   = $012f          PRINT USING : Exponent-Flag (Feld)
00438  0000            etof   = $0130          PRINT USING : Schalter
00439  0000            cform  = $0131          PRINT USING : Zeichen-Zähler
00440  0000            sno    = $0132          PRINT USING : Vorzeichennummer
00441  0000            blfd   = $0133          PRINT USING : Blank/Stern-Flag
00442  0000            begfd  = $0134          PRINT USING : Zeiger auf Feld
00443  0000            lfor   = $0135          PRINT USING : Länge Formatstring
00444  0000            endfd  = $0136
00445  0000
00446  0000  ===> weitere COMMON RAM Labels <===
00447  0000
00448  0000            lolink = $01fc          Low-Byte Vorbelegung Zeilen-Link
00449  0000            hilink = $01fd          High-Byte Vorbelegung Zeilen-Link
00450  0000            buf    = $0200          Eingabe-Puffer
00451  0000
00452  0000            _bufln = 161            Länge Eingabepuffer
00453  0000
00454  0000            stashx = $02b9          STAVEC
00455  0000            efnvec = $02fc          Vector Aufruf user-def. Funktion
                                               ($4c78)
00456  0000            ierror = $0300          ind. Fehlerbehandlung ($4d3f)
00457  0000            imain  = $0302          ind. BASIC Eingabeschleife ($4dc6)
00458  0000            icrnch = $0304          ind. Wandlung Text/Tokens ($430d)
00459  0000            iqplop = $0306          ind. LIST/HELP Wandlung Tokens/Text
                                               ($5151)
00460  0000            igone  = $0308          ind. Routine zu Token (ac) aufrufen
                                               ($4aa2)
00461  0000            ieval  = $030a          ind. folg. Ausdruck auswerten
                                               ($78da)
```

```
00462  0000            iesclk = $030c          ind. Wandlg. Text/user-def. Token
                                                ($4321)
00463  0000            iescpr = $030e          ind. Wandlg. user-def. Token/Text
                                                ($51cd)
00464  0000            iescex = $0310          ind. Routine zu user-def. Token
                                                aufr. ($4ba9)
00465  0000            keyd   = $034a          Tast.puffer
00466  0000            chrget = $0380          nächstes Zeichen nach ac
00467  0000            chrgot = $0386          letztes Zeichen erneut nach ac
00468  0000            qnum   = $0390          Test ac numerisch (CARRY=0)
00469  0000            iram0  = $039f          LDA (ac),y aus PGM-RAM
00470  0000            iram1  = $03ab          LDA (ac),y aus VAR-RAM
00471  0000            ind1_1 = $03b7          LDA (index1),y aus VAR-RAM
00472  0000            ind2_0 = $03c0          LDA (index2),y aus PGM-RAM
00473  0000            indtxt = $03c9          LDA (txtptr),y aus PGM-RAM
00474  0000            zero   = $03d2          Konstante Null
00475  0000            curbnk = $03d5          akt. Bank
00476  0000            tmdes1 = $03d6          ADDRL Desc.
00477  0000            tmdes2 = $03d8          ADDRL Desc.
00478  0000            finbnk = $03da          Bank chrget aus String
00479  0000            savsiz = $03db          Low-Byte Breite SHAPE Bereich
00480  0000            bits   = $03df          FAC Overflow Bit
00481  0000            sptmp1 = $03e0
00482  0000            sptmp2 = $03e1
00483  0000            fgbg   = $03e2          VIC Vorder-/Hintergrund Farbe
00484  0000            fgmc1  = $03e3          Zusatzfarbe 1
00485  0000            comtop = $0400          Obergrenze COMMON RAM
00486  0000            stktop = $0800          Obergrenze BASIC-Stapel
00487  0000
00488  0000
00489  0000            _lnfor = 18             Anzahl Stapel-Bytes für FOR/NEXT
                                                Schleife
00490  0000            _fords = 1              Zeiger Descriptor FOR Var. in Stapel
00491  0000            _step  = 3              Zeiger STEP Wert in Stapel
00492  0000            _stpsg = 8              Zeiger Vorzeichen STEP Wert in
                                                Stapel
00493  0000
00494  0000            _lngos = 5              Anzahl Stapel-Bytes für GOSUB
00495  0000
00496  0000            stkbot = $09ff          Untergrenze BASIC-Stapel
00497  0000
00498  0000  ===> Labels für SPRITES / GRAPHIK <===
00499  0000
00500  0000            rstvec = $0a00          Restart-Vector
00501  0000            initst = $0a04          Flag BASIC-IRQ: bit0 = 1 (OP-Syst.)
00502  0000            slpcnt = $0a1d          Timer für SLEEP
00503  0000            split  = $0a34          VIC Rasterwert geteilter BS
00504  0000
00505  0000            sprbas = $0e00          Sprite Basis Adresse
00506  0000
00507  0000            pkylen = $1000          Tab. Längen Fkt.Tasten Texte
00508  0000            pkydef = $100a          Puffer für Fkt.tasten-Befehle
00509  0000            dosstr = $1100          DOS Kommando String Puffer
00510  0000            vwork  = $1131
00511  0000            xypos  = $1131          Ord. low PIXEL CURS.
00512  0000
00513  0000            ypos   = $1133
```

```
00514  0000
00515  0000              xdest  = $1135
00516  0000              ydest  = $1137
00517  0000
00518  0000              xyabs  = $1139
00519  0000              xabs   = $1139
00520  0000
00521  0000              yabs   = $113b
00522  0000              xysgn  = $113d
00523  0000              ysgn   = $113f
00524  0000              fct    = $1141
00525  0000              lesser = $1147
00526  0000              greatr = $1148
00527  0000              angsgn = $1149          Quadrant
00528  0000              sinval = $114a
00529  0000              cosval = $114c
00530  0000              angcnt = $114e
00531  0000
00532  0000              xcord1 = $1150
00533  0000              xcircl = $1150
00534  0000
00535  0000              ycord1 = $1152
00536  0000              ycircl = $1152
00537  0000              strsz  = $1153          Stringlänge
00538  0000
00539  0000              xdist1 = $1154
00540  0000              boxang = $1154
00541  0000              gettyp = $1154
00542  0000              xradus = $1154
00543  0000
00544  0000              strptr = $1155          Stringzeiger
00545  0000              xcount = $1156
00546  0000              yradus = $1156
00547  0000
00548  0000              newbyt = $1157
00549  0000              ycount = $1158
00550  0000              rotang = $1158
00551  0000              xsize  = $1159
00552  0000              ydist2 = $115a
00553  0000              ysize  = $115b
00554  0000
00555  0000              xcord2 = $115c
00556  0000              xcord2 = $115c
00557  0000              disend = $115c
00558  0000
00559  0000              xsave  = $115d
00560  0000              ycord2 = $115e
00561  0000              colcnt = $115e
00562  0000
00563  0000              rowcnt = $115f          BS Zeile CHAR
00564  0000              xrcos  = $1160
00565  0000              strcnt = $1160
00566  0000
00567  0000              bitidx = $1161          Bitoffset
00568  0000              charom = $1168
00569  0000              bitcnt = $1169          Bitzähler
00570  0000              scalem = $116a          SCALE Flag
00571  0000              width  = $116b          Strichstärke (0-1)
```

```
00572  0000            filflg = $116c          Flag 1=Ausmalen, 0=nicht
00573  0000            bitmsk = $116d
00574  0000            numcnt = $116e          Zeichenzähler
00575  0000            trcflg = $116f          TRACE-Flag (0=aus, ff=ein)
00576  0000            rnstln = $1170          Low-Byte Startzeile neu
00577  0000            rstep  = $1172          Low-Byte Schrittweite
00578  0000            t3     = $1174
00579  0000            t4     = $1175
00580  0000            t5     = $1176
00581  0000            vtemp3 = $1177
00582  0000            vtemp4 = $1178
00583  0000            vtemp5 = $1179
00584  0000            adray1 = $117a          ADDRL Vektor Wandlung Gleitk./Int.
00585  0000            adray2 = $117c          ADDRL Vektor Wandlung Int./Gleitk.
00586  0000            sspeed = $117e          Konstante Schritte / BASIC-IRQ
00587  0000            scnst  = $117f          Zähler für Schritte /BASIC-IRQ
00588  0000            squad  = $1180          Quadrant Sprite
00589  0000            sxswt  = $1181          Low-Byte Schrittweite x-Richtung
00590  0000            syswt  = $1183          Low-Byte Schrittweite y-Richtung
00591  0000            sabsz  = $1185          Low-Byte akt. Abszisse (x-Wert)
00592  0000            sord   = $1187          Low-Byte akt. Ordinate (y-Wert)
00593  0000            postab = $11d6          Abszisse Sprite
00594  0000            spmsb  = $11e6          MSB-Position Sprite7-0
00595  0000            sscoll = $11e7          Flags Sprite Kollision mit Sprite7-0
00596  0000            sbcoll = $11e8          Flags Hintergrund Kollision mit
                                                Sprite7-0
00597  0000            lpabsz = $11e9          Kopie Light-Pen, x-Wert (Abszisse)
00598  0000            lpord  = $11ea          Kopie Light-Pen, y-Wert (Ordinate)
00599  0000            upplow = $11eb          ADDRH Char.gen. Klein / Groß
00600  0000            upgrap = $11ec          ADDRH Char.gen. Groß / Graphik
00601  0000            tdossa = $11ed
00602  0000            oldlin = $1200          Low-Byte vorige Zeilennummer (stop)
00603  0000            oldtxt = $1202          ADDRL Zeiger letzter BASIC-Befehl
00604  0000            pufill = $1204          PRINT USING: Füllzeichen
00605  0000            pucoma = $1205
00606  0000            pudot  = $1206
00607  0000            pumony = $1207
00608  0000            errnum = $1208          Nummer Fehlertext
00609  0000            errlin = $1209          Low-Byte Zeilennummer für RESUME
00610  0000            trapno = $120b          Low Zeilennr. TRAP Routine
00611  0000            tmptrp = $120d          High Zeilennr. TRAP Routine bei TRAP
00612  0000            errtxt = $120e          ADDRL txtptr bei Fehler
00613  0000            txttop = $1210          ADDRL Obergrenze akt. PGM
00614  0000            mxmem0 = $1212          ADDRL Obergr. PGM RAM
00615  0000            tmptxt = $1214          Kopie ADDRL chrget-Zeiger
00616  0000            tmplin = $1216          Kopie Low-Byte Zeilennummer
00617  0000            usrpok = $1218          jmp-Opcode (USR-Routine)
00618  0000            rndx   = $121b          GK Variable RND
00619  0000
00620  0000  ===> Labels für SOUND und BASIC IRQ <===
00621  0000
00622  0000            crcseg = $1220          Segmentwinkel
00623  0000            tmprat = $1222          Spiel Tempo
00624  0000            voiclo = $1223          Tab. Low-Bytes Tondauer (1-3)
00625  0000            voichi = $1224          Tab. High-Bytes Tondauer (1-3)
00626  0000            ntime  = $1229          Low-Byte Tondauer Konst.
00627  0000            octave = $122b          OCTAVE Nummer 0-6
00628  0000            sharp  = $122c          Flag +/- Halbton
```

```
00629  0000            pitch  = $122d          Low-Byte Freq.Konst.
00630  0000            voice  = $122f          VOICE Nummer -1
00631  0000            wave   = $1230          Tab. Hüllk.werte VOICE 1-3
00632  0000            dnote  = $1233          Flag für punktierte Note
00633  0000            fltsav = $1234          2. Kopie SID Reg. 21 (FCLOW)
00634  0000            fltflg = $1238          Bit Muster für Paß Charakteristik
00635  0000            nibble = $1239          Puffer oberes/unteres Halb Byte
00636  0000            tonnum = $123a          Nummer HÜLLKURVE 0-9
00637  0000            tonval = $123b          ATTACK/DECAY akt. HÜLLKURVE
00638  0000            parcnt = $123e
00639  0000            atktab = $123f          Tab. ATTACK/DECAY akt. HÜLLKURVEN
00640  0000            sustab = $1249          Tab. SUSTAIN/RELEASE akt. HÜLLK.
00641  0000            wavtab = $1253          Tab. WELLENFORMEN akt. HÜLLKURVEN
00642  0000            pulslo = $125d          Tab. Low-Bytes PULSBREITE akt. HÜLLK.
00643  0000            pulshi = $1267          Tab. High-Byte PULSBREITE akt. HÜLLK.
00644  0000            flters = $1271          Kopie SID Reg. 21 (FCLOW)
00645  0000            oldvol = $1275          letzter Wert SID VOL/FILT
00646  0000            tripfg = $1276          Flag Bearbeitung Sprite/Sprite Koll.
00647  0000            sbcflg = $1277          Flag Bearbeitung Sprite/Hintergr.
                                               Koll.
00648  0000            lpnflg = $1278          Flag Bearbeitung Light-Pen
00649  0000            intadl = $1279          Tab. Low-Byte Zeilennr. COLISN
00650  0000            intadh = $127c          Tab. High-Byte Zeilennr. COLISN
00651  0000
00652  0000            intval = $127f          Flag für Bearbeitung COLLISION
00653  0000
00654  0000   7.......  1=COLLISION wird bearbeitet
00655  0000   .6543...  unbenutzt
00656  0000   .....2..  1=Zeilennr. für Light-Pen definiert
00657  0000   ......1.  1=Zeilennr. für Sprite/Hintergr. Koll. def.
00658  0000   .......0  1=Zeilennr. für Sprite/Sprite Koll. def.
00659  0000
00660  0000            coltyp = $1280          COLLISION Typ (0-2)
00661  0000            snvoic = $1281          VOICE 0-2
00662  0000            sndtl1 = $1282          Low-Byte Tondauer VOICE 1
00663  0000            sndth1 = $1285          High-Byte Tondauer VOICE 1
00664  0000            sndth2 = $1286          High-Byte Tondauer VOICE 2
00665  0000            sndth3 = $1287          High-Byte Tondauer VOICE 3
00666  0000            sdmxl1 = $1288          Low-Byte Grenzfreq.1 VOICE 1
00667  0000            sdmxh1 = $128b          High-Byte Grenzfreq.1 VOICE 1
00668  0000            sdmnl1 = $128e          Low-Byte Grenzfreq.2 VOICE 1
00669  0000            sdmnh1 = $1291          High-Byte Grenzfreq.2 VOICE 1
00670  0000            sddir1 = $1294          Richtung VOICE 1
00671  0000            sdspl1 = $1297          Low-Byte Schrittweite VOICE 1
00672  0000            sdsph1 = $129a          High-Byte Schrittweite VOICE 1
00673  0000            sdfql1 = $129d          Low-Byte Frequenz VOICE 1
00674  0000            sdfqh1 = $12a0          High-Byte Frequenz VOICE 1
00675  0000            tmtiml = $12a3          Low-Byte Tondauer
00676  0000            tmtimh = $12a4          High-Byte Tondauer
00677  0000            tmmaxl = $12a5          Low-Byte Grenzfrequenz 1
00678  0000            tmmaxh = $12a6          High-Byte Grenzfrequenz 1
00679  0000            tmminl = $12a7          Low-Byte Grenzfrequenz 2
00680  0000            tmminh = $12a8          High-Byte Grenzfrequenz 2
00681  0000            tmpdir = $12a9          Richtung 0-2
00682  0000            tmstpl = $12aa          Low-Byte Schrittweite
00683  0000            tmstph = $12ab          High-Byte Schrittweite
00684  0000            tmfrql = $12ac          Low-Byte Frequenz
00685  0000            tmfrqh = $12ad          High-Byte Frequenz
```

```
00686  0000            tmpul  = $12ae          Low-Byte PULSBREITE
00687  0000            tmpulh = $12af          High-Byte PULSBREITE
00688  0000            tmwave = $12b0          WELLENFORM
00689  0000            tmpot1 = $12b1          Flag Paddle 1/2 bzw. 3/4
00690  0000            tmpot2 = $12b2          A/D Wandler Wert
00691  0000            wintmp = $12b3          WINDOW Spalte BS Fenster li.ob.
00692  0000            savram = $12b7
00693  0000            defmod = $12fa          Modus SPRDEF
00694  0000            lincnt = $12fb          Zeilen Zähler
00695  0000            spnumb = $12fc          Sprite Nummer -1
00696  0000            birqfg = $12fd          BASIC-IRQ Flag (<>0 = in Arbeit)
00697  0000
00698  0000            rambot = $1c00          Untergrenze PGM RAM
00699  0000            baswrk = $1c01          Untergrenze PGM
00700  0000
00701  0000            colrml = $1c00          COLOR RAM low
00702  0000            gpbase = $2000          GRAPHIC RAM
00703  0000            gbasic = $4000          Untergrenze BASIC mit Graphik
00704  0000
00705  0000
00706  0000
00707  0000  ===> Definition ROM-Labels BS-Editor <===
00708  0000
00709  0000            monit  = $b000          BASIC-Statement MONITOR ($fa)
00710  0000            jprint = $c00c          BS Ausgabe ac
00711  0000            jwind  = $c02d          BS Fenster definieren
00712  0000            ldtb2  = $c033          Tab. ADDRL BS Zeilen
00713  0000            ldtb1  = $c04c          Tab. ADDRH BS Zeilen
00714  0000            chrgug = $d000          Basisadr. Char.gen. Groß / Graphik
00715  0000            chrglu = $d800          Basisadr. Char.gen. Klein / Groß
00716  0000            colrmh = $d800          COLOR RAM high
00717  0000
00718  0000  ===> Definitionen ROM-Labels Operating-System <===
00719  0000
00720  0000            mmucr  = $ff00          MMU 8722 Config. Reg. (Kopie)
00721  0000            lcra   = $ff01          64KB PGM-RAM einschalten
00722  0000            lcrb   = $ff02          64KB VAR-RAM einschalten
00723  0000            lcrc   = $ff03          ROM / PGM-RAM einschalten
00724  0000            lcrd   = $ff04          ROM / VAR-RAM einschalten
00725  0000            kclsal = $ff4a          alle Files von Gerät in ac schließen
                                                (Sprungv.)
00726  0000            kc64md = $ff4d          auf C64-Modus umschalten (Sprungv.)
00727  0000            kdmacl = $ff50          DMA-Anforderung ext. RAM zulassen
                                                (Sprungv.)
00728  0000            kbotcl = $ff53          Boot-Programm von Diskette laden
                                                (Sprungv.)
00729  0000            kphenx = $ff56          Aufruf Kaltstartroutinen
                                                Funktionskarte (Sprungv.)
00730  0000            klkupl = $ff59          Geräte- & Sek.-Adr. über la (ac)
                                                suchen (Sprungv.)
00731  0000            lkups  = $ff5c          Geräte-Adr. über Sek.Adr. (yr)
                                                suchen (Sprungv.)
00732  0000            kswapr = $ff5f          Umschalten 40/80 Zeichen Bildschirm
                                                (flip) (Sprungv.)
00733  0000            kpfkey = $ff65          Funktionstasten programmieren
                                                (Sprungv.)
00734  0000            kstbnk = $ff68          ac nach RAM-Bank, xr nach
                                                Filename-Bank (Sprungv.)
```

```
00735  0000            kjsrfr = $ff6e          JSR in andere Bank, RTS in akt. Bank
                                                (Sprungv.)
00736  0000            kjmpfr = $ff71          JMP in andere Bank (Sprungv.)
00737  0000            kifetv = $ff74          LDA (FETVEC),y von Bank in xr
                                                (Sprungv.)
00738  0000            kistav = $ff77          STA (STAVEC),y in Bank in xr
                                                (Sprungv.)
00739  0000            kicmpv = $ff7a          CMP (CMPVEC),y mit Bank in xr
                                                (Sprungv.)
00740  0000            kprimm = $ff7d          folg. Text ausg./überspringen
                                                (Sprungv.)
00741  0000            kstmsg = $ff90          Flag setzen Ausgabe/Unterdr.
                                                OPSYS-Meldung (Sprungv.)
00742  0000            kreads = $ffb7          RS-/System-Status nach ac (Sprungv.)
00743  0000            kstlfs = $ffba          ac, xr, yr nach la, fa, sa
                                                (Sprungv.)
00744  0000            ksetnm = $ffbd          ac, xr/yr nach fnlen, fnadr/+1
                                                (Sprungv.)
00745  0000            kopen  = $ffc0          log. File üb. Vektor öffnen, Ausgabe
                                                fn (sprungv.)
00746  0000            kclose = $ffc3          log. File (la in ac) schließen
                                                (Sprungv.)
00747  0000            kchkin = $ffc6          Eingabekanal üb. Vektor öffnen
                                                (Sprungv.)
00748  0000            kckout = $ffc9          Ausgabe-Kanal üb. Vektor öffnen
                                                (Sprungv.)
00749  0000            kclrch = $ffcc          ind. I/O-Kanäle schließen (Sprungv.)
00750  0000            kbasin = $ffcf          ind. Zeich. akt. Kanal nach ac
                                                (Sprungv.)
00751  0000            kbsout = $ffd2          ind. Ausg. ac auf akt. Kanal
                                                (Sprungv.)
00752  0000            kloads = $ffd5          load/verify (ac) ab Adresse in xr/yr
                                                (Sprungv.)
00753  0000            ksaves = $ffd8          save ab ac bis xr, yr (Sprungv.)
00754  0000            ksttim = $ffdb          System-Uhr setzen (Sprungv.)
00755  0000            krdtim = $ffde          System-Uhr nach ac, xr, yr
                                                (Sprungv.)
00756  0000            kstop  = $ffe1          Abfrage Stopp-Taste, dann Ersatz-I/O
                                                (Sprungv.)
00757  0000            kgetin = $ffe4          Zeichen üb. Vektor vom akt. Kanal
                                                nach ac (Sprungv.)
00758  0000            kclall = $ffe7          off. Files :=0, Ersatz-I/O
                                                (Sprungv.)
00759  0000            kscror = $ffed          Bildschirmorganisation lesen
                                                (Sprungv.)
00760  0000            kplot  = $fff0          Cursor-Pos. (rel. BS-Fenster)
                                                les./schreib. (Sprungv.)
00761  0000
00762  0000
00763  0000                    .end
00764  0000                    .lib basic.cpu
00765  0000  put "@basic.cpu"
```

```
00767  0000
00768  0000
00769  0000  ===> CPU 8502 I/O-Port <===
00770  0000
00771  0000  ---- CPU 8502 Port-Datenrichtungs-Register ----
00772  0000
00773  0000            d8502  = $00            CPU 8502 Port-Datenrichtungs-
                                                Register
00774  0000
00775  0000   x0101111  0 = Input, 1 = Output (s. Daten-Reg.)
00776  0000
00777  0000  ---- CPU 8502 Port-Daten-Register ----
00778  0000
00779  0000            r8502  = $01            CPU 8502 Port-Daten-Register
00780  0000
00781  0000   7.......  unbenutzt
00782  0000
00783  0000   .6......  Status CAPS LOCK/DIN Taste
00784  0000
00785  0000   x0xxxxxx  Taste gedrückt
00786  0000   x1xxxxxx  Taste nicht gedrückt
00787  0000
00788  0000   ..5.....  Steuerung Kassettenmotor
00789  0000
00790  0000   xx0xxxxx  Motor ein
00791  0000   xx1xxxxx  Motor aus
00792  0000
00793  0000   ...4....  Eingang Taste Kassettenrek.
00794  0000
00795  0000   xxx0xxxx  Taste an Recorder gedrückt
00796  0000   xxx1xxxx  Taste an Recorder nicht gedrückt
00797  0000
00798  0000   ....3...  Kassette Schreibleitung
00799  0000
00800  0000   xxxx0xxx  serielle Schreibleitung 0
00801  0000   xxxx1xxx  serielle Schreibleitung 1
00802  0000
00803  0000   .....2..  Auswahl I/O / ROM $d000-$dfff (C 64)
00804  0000
00805  0000   xxxxx0xx  Character Generator ROM ein $d000-$dfff
00806  0000   xxxxx1xx  I/O-Chips C 64 ein $d000-$dfff
00807  0000
00808  0000   ......1.  Auswahl KERNAL ROM / RAM $e000-$ffff (C 64)
00809  0000
00810  0000   xxxxxx0x  8 KByte RAM ein $e000-$ffff
00811  0000   xxxxxx1x  KERNAL ROM C 64 ein $d000-$dfff
00812  0000
00813  0000   .......0  Auswahl BASIC ROM / RAM $a000-$bfff (C 64)
00814  0000
00815  0000   xxxxxxx0  MULTICOLOR RAM ein $d800-$dbff
00816  0000   xxxxxxx1  BASIC ROM C 64 ein $a000-$bfff
00817  0000
00818  0000
00819  0000                    .end
00820  0000                    .lib basic.vic
00821  0000  put "@basic.vic"
```

```
00823  0000
00824  0000
00825  0000  ===> VIC 8564 Video Controller <===
00826  0000
00827  0000   unbenutzte Bits erscheinen beim Lesen immer als 1.
00828  0000
00829  0000  Reg.    Bits
00830  0000  ==============
00831  0000
00832  0000             vic    = $d000          Basisadresse VIC 8564
00833  0000
00834  0000    0   76543210  Sprite0, Low-Byte x-Wert (Abszisse)
00835  0000    1   76543210  Sprite0, Low-Byte y-Wert (Ordinate)
00836  0000
00837  0000    2
00838  0000    .   Register 2-15 enthalten entsprechend die
00839  0000    .   Low-Bytes der Koordinaten von Sprite 1-7.
00840  0000   15
00841  0000
00842  0000   16   76543210  MSB (=Bit8) Abszisse Sprite7-0
00843  0000
00844  0000             viccr1 = $d011          VIC 8564 Control-Register 1
00845  0000
00846  0000   17   7.......  MSB (=Bit8) Raster-Wert Reg. 18
00847  0000        .6......  Extendet Color Text (0=aus, 1=ein)
00848  0000        ..5.....  Bit Map Mode (0=aus, 1=ein)
00849  0000        ...4....  Blank Screen to Color Border (0=Blank)
00850  0000        ....3...  24/25 Zeilen BS (0=24, 1=25)
00851  0000        .....210  Soft-Scroll vertikal, Dot-Bits 0-7
00852  0000
00853  0000   18   76543210  Low-Byte Raster-Vorgabe für Raster IRQ (WRITE)
00854  0000                  Low-Byte akt. Raster-Wert (READ)
00855  0000                  MSB (=Bit8) in Reg. 17, Bit7
00856  0000
00857  0000             viclpx = $d013          VIC 8564 Light-Pen x-Wert
00858  0000
00859  0000   19   76543210  Light-Pen, x-Wert (Abszisse)
00860  0000
00861  0000             viclpy = $d014          VIC 8564 Light-Pen y-Wert
00862  0000
00863  0000   20   76543210  Light-Pen, y-Wert (Ordinate)
00864  0000
00865  0000             vicmoe = $d015          VIC 8564 Sprite Enable Register
00866  0000
00867  0000   21   76543210  Sprite7-0 (1=ein)
00868  0000
00869  0000   22   76......  unbenutzt
00870  0000        ..5.....  immer 0 setzen (reserviert für Weiterentw.)
00871  0000        ...4....  Priorität (0=aus, 1=ein)
00872  0000        ....3...  38/40 Spalten/Zeile (0=38, 1=40)
00873  0000        .....210  Soft-Scroll horizontal, Dot-Bits 0-7
00874  0000
00875  0000   23   76543210  Sprite7-0 vert. verdoppeln (1=ein)
00876  0000
00877  0000   24   7654....  Adreßbits 13-10 Basisadresse BS RAM
00878  0000        ....321.  Adreßbits 13-11 Basisadresse Chargen. ROM
00879  0000        .......0  unbenutzt
00880  0000
```

```
00881  0000            vicifr = $d019          VIC 8564 Interrupt Flag-Register
00882  0000
00883  0000   25   7.......  Interrupt-Flag (1=VIC IRQ)
00884  0000        .654....  unbenutzt
00885  0000        ....3...  Light-Pen IRQ (1=IRQ)
00886  0000        .....2..  Sprite/Sprite Kollision (1=IRQ)
00887  0000        ......1.  Sprite/Hintergrund Kollision (1=IRQ)
00888  0000        .......0  Raster-Vergleich IRQ (1=IRQ)
00889  0000
00890  0000   26   7654....  unbenutzt
00891  0000        ....3...  Light-Pen IRQ (1=IRQ enabled)
00892  0000        .....2..  Sprite/Sprite Kollision (1=IRQ enabled)
00893  0000        ......1.  Sprite/Hintergrund Kollision (1=IRQ enabled)
00894  0000        .......0  Raster-Vergleich IRQ (1=IRQ enabled)
00895  0000
00896  0000   27   76543210  Priorität für Sprite7-0 (1=ein)
00897  0000
00898  0000            vicmcs = $d01c          VIC 8564 Multi-Color-Modus Sprite7-0
00899  0000
00900  0000   28   76543210  Multi-Color-Modus für Sprite7-0 (1=ein)
00901  0000
00902  0000   29   76543210  Sprite7-0 hor. verdoppeln (1=ein)
00903  0000
00904  0000            vicssc = $d01e          VIC 8564 Sprite-Kollision mit
                                               Sprite7-0
00905  0000
00906  0000   30   76543210  Sprite-Kollision mit Sprite7-0 (1=ein)
00907  0000
00908  0000   31   76543210  Hintergrund-Kollision mit Sprite7-0 (1=ein)
00909  0000
00910  0000            vicbrd = $d020          VIC 8564 Rahmenfarbe
00911  0000
00912  0000   32   7654....  unbenutzt
00913  0000        ....3210  Rahmenfarbe (0-15)
00914  0000
00915  0000            vicbkg = $d021          VIC 8564 Hintergrundfarbe
00916  0000
00917  0000   33   7654....  unbenutzt
00918  0000        ....3210  Hintergrundfarbe #0 (0-15)
00919  0000
00920  0000   34   7654....  unbenutzt
00921  0000        ....3210  Hintergrundfarbe #1 (0-15)
00922  0000
00923  0000   35   7654....  unbenutzt
00924  0000        ....3210  Hintergrundfarbe #2 (0-15)
00925  0000
00926  0000   36   7654....  unbenutzt
00927  0000        ....3210  Hintergrundfarbe #3 (0-15)
00928  0000
00929  0000            vicmc0 = $d025          VIC 8564 Sprite Multicolor #0
00930  0000
00931  0000   37   7654....  unbenutzt
00932  0000        ....3210  Sprite Multicolor #0 (0-15)
00933  0000
00934  0000            vicmc1 = $d026          VIC 8564 Sprite Multicolor #1
00935  0000
00936  0000   38   7654....  unbenutzt
00937  0000        ....3210  Sprite Multicolor #1 (0-15)
```

```
00938  0000
00939  0000              vicspc = $d027          VIC 8564 Farbe Sprite0
00940  0000
00941  0000   39   7654....  unbenutzt
00942  0000        ....3210  Farbe Sprite0 (0-15)
00943  0000
00944  0000   40
00945  0000    .   Register 40-46 enthalten entsprechend die
00946  0000    .   Farben (0-15) der Sprites 1-7
00947  0000   46
00948  0000
00949  0000   47   7654....  unbenutzt
00950  0000        ....3210  VIC Tastatur-Port (K3-K0)
00951  0000
00952  0000              vicspd = $d030          VIC 8564 Speed-Register
00953  0000
00954  0000   48   7654321.  unbenutzt
00955  0000        .......0  Flag CPU-Takt (0 = 1 MHz, 1 = 2 MHz)
00956  0000
00957  0000
00958  0000                    .end
00959  0000                    .lib basic.sid
00960  0000  put"@basic.sid"
```

```
00961  0000
00962  0000
00963  0000
00964  0000  ===> SID 6581 Sound Interface Device <===
00965  0000
00966  0000            sid    = $d400          Basisadresse SID 6581
00967  0000
00968  0000   ---- Frequenz ----
00969  0000
00970  0000            freqlo  = 0             Offset Low-Byte Frequenz
00971  0000
00972  0000   7643210    Low-Byte Frequenz Stimme 1
00973  0000
00974  0000            freqhi  = 1             Offset High-Byte Frequenz
00975  0000
00976  0000   76543210   High-Byte Frequenz Stimme 1
00977  0000
00978  0000  ----Pulsbreite ----
00979  0000
00980  0000            pulsef  = 2             Offset Low-Byte Pulsbreite
00981  0000
00982  0000   76543210  Low-Byte Pulsbreite Stimme 1
00983  0000
00984  0000            pulsec  = 3             Offset High-Byte Pulsbreite
00985  0000
00986  0000   7654....  unbenutzt
00987  0000   ....3210  High-Byte Nybble Pulsbreite Stimme 1
00988  0000
00989  0000  ---- Wellenform ----
00990  0000
00991  0000            oscctl  = 4             Offset Wellenform
00992  0000
00993  0000   7.......  1 = Zufallsgenerator ein
00994  0000   .6......  1 = Rechteckschwingung ein
00995  0000   ..5.....  1 = Sägezahnschwingung ein
00996  0000   ...4....  1 = Dreieckschwingung ein
00997  0000   ....3...  1 = Oszillator 1 aus
00998  0000   .....2..  1 = Stimme 1 mit 3 ringmodulieren
00999  0000   ......1.  1 = Stimme 1 mit 3 synchronisieren
01000  0000   .......0  1 = Start Att/Dec/Sus, 0 = Start Rel.
01001  0000
01002  0000  ---- Attack / Decay ----
01003  0000
01004  0000            atkdcy  = 5             Offset Attack/Decay
01005  0000
01006  0000   7654....  Attack  - Zeit (0-15)
01007  0000   ....3210  Decay   - Zeit (0-15)
01008  0000
01009  0000  ---- Sustain / Release ----
01010  0000
01011  0000             susrel = 6             Offset Sustain/Release
01012  0000
01013  0000   7654....  Sustain - Zeit (0-15)
01014  0000   ....3210  Release - Zeit (0-15)
01015  0000
01016  0000
01017  0000  ---- Offsets Stimmen (VOICE)  ----
01018  0000
```

```
01019  0000                    osc1 = 0          Offset VOICE 1
01020  0000                    osc2 = 7          Offset VOICE 2
01021  0000                    osc3 = 14         Offset VOICE 3
01022  0000
01023  0000  ---- Stimme 2 ----
01024  0000
01025  0000  Mit den Registern 7-13 ($d407-$d40d) wird Stimme 2 analog
01026  0000  gehandhabt, dabei gilt für Register 11 (Control-Register):
01027  0000
01028  0000  ---- sid+osc2-oscctl ----
01029  0000
01030  0000   .....2..  1 = Stimme 2 mit 1 ringmodulieren
01031  0000   ......1.  1 = Stimme 2 mit 1 synchronisieren
01032  0000
01033  0000  ---- Stimme 3 ----
01034  0000
01035  0000  Mit den Registern 14-20 ($d40e-$d414) wird Stimme 3 analog
01036  0000  gehandhabt, dabei gilt für Register 18 (Control-Register):
01037  0000
01038  0000  ---- sid+osc3+oscctl
01039  0000
01040  0000   .....2..  1 = Stimme 3 mit 2 ringmodulieren
01041  0000   ......1.  1 = Stimme 3 mit 2 synchronisieren
01042  0000
01043  0000  ---- Filter-Grenzfrequenz ----
01044  0000
01045  0000            fclow  = 21           Offset Low-Byte Filter Grenzfrequenz
01046  0000
01047  0000   76543...  unbenutzt
01048  0000   .....210  Low-Nibble Filter Grenzfrequenz (0-7)
01049  0000
01050  0000            fchi   = 22           Offset High-Byte Filter Grenzfrequenz
01051  0000
01052  0000   76543210  ;High-Byte Filter Grenzfrequenz (0-255)
01053  0000
01054  0000  ---- Resonanz und Filter-Eingangsschalter ----
01055  0000
01056  0000            resnce = 23           Offset Filter Schaltung / Resonanz
01057  0000
01058  0000   7654....  Filter-Resonanz 0-15
01059  0000   ....3...  1 = ext. Eingang über Filter
01060  0000   .....2..  1 = Stimme 3     über Filter
01061  0000   ......1.  1 = Stimme 2     über Filter
01062  0000   .......0  1 = Stimme 1     über Filter
01063  0000
01064  0000  ---- Filter-Modus und Lautstärke ----
01065  0000
01066  0000            fltvol = 24           Offset Filter Art / Lautstärke
01067  0000
01068  0000   7.......  1 = Stimme 3 abgeschaltet
01069  0000   .6......  1 = Filter Hochpaß ein
01070  0000   ..5.....  1 = Filter Bandpaß ein
01071  0000   ...4....  1 = Filter Tiefpaß ein
01072  0000   ....3210  1 = Lautstärke (0-15)
01073  0000
01074  0000  ---- A/D Wandler 1 ----
01075  0000
01076  0000            potx   = 25           Offset A/D Wandler 1
```

```
01077  0000
01078  0000  ---- A/D Wandler 2 ----
01079  0000
01080  0000            poty   = 26            Offset A/D Wandler 2
01081  0000
01082  0000  ---- Rauschgenerator ----
01083  0000
01084  0000            random = 27            Offset Rauschgenerator
01085  0000
01086  0000  ---- Hüllkurvengenerator ----
01087  0000
01088  0000            cenv3  = 28            Offset HÜLLKURVE lesen
01089  0000
01090  0000
01091  0000                    .end
01092  0000                    .lib basic.mmu
01093  0000  put"@basic.mmu"
```

```
01094  0000
01095  0000
01096  0000
01097  0000  ===> MMU 8722 Memory Management Unit <===
01098  0000
01099  0000  ---- Configuration-Register ----
01100  0000
01101  0000            mmucrl = $d500          MMU 8722 Configuration-Register
01102  0000
01103  0000   76......  Auswahl der 64 K RAM-Bank:
01104  0000
01105  0000   00xxxxxx  RAM Bank 0
01106  0000   01xxxxxx  RAM Bank 1
01107  0000   10xxxxxx  RAM Bank 2 (wenn nicht vorhanden: Bank 0)
01108  0000   11xxxxxx  RAM Bank 3 (wenn nicht vorhanden: Bank 1)
01109  0000
01110  0000   ..54....  Auswahl RAM/ROM Adreßbereich $c000 - $ffff
01111  0000
01112  0000   xx00xxxx  Kernal ROM
01113  0000   xx01xxxx  Internal Function ROM
01114  0000   xx10xxxx  External Function ROM
01115  0000   xx11xxxx  RAM Bank, gewählt mit Bit7/6
01116  0000
01117  0000   ....32..  Auswahl RAM/ROM Adreßbereich $8000 - $bfff
01118  0000
01119  0000   xxxx00xx  Teil BASIC ROM
01120  0000   xxxx01xx  Internal Function ROM
01121  0000   xxxx10xx  External Function ROM
01122  0000   xxxx11xx  RAM Bank, gewählt mit Bit7/6
01123  0000
01124  0000   ......1.  Auswahl RAM/ROM Adreßbereich $4000 - $7fff
01125  0000
01126  0000   xxxxxx0x  Teil BASIC ROM
01127  0000   xxxxxx1x  RAM Bank, gewählt mit Bit7/6
01128  0000
01129  0000   .......0  Auswahl RAM/ROM-I/O Adreßbereich $d000 - A$dfff
01130  0000
01131  0000   xxxxxxx0  I/O-Chips
01132  0000   xxxxxxx1  RAM/ROM/Char.gen., gewählt mit Bit5/4
01133  0000
01134  0000
01135  0000  ---- Preconfiguration- / Load Configuration-Register ----
01136  0000
01137  0000            pcra    = $d501         Preconfiguration-Register A
01138  0000
01139  0000            pcrb    = $d502         Preconfiguration-Register B
01140  0000
01141  0000            pcrc    = $d503         Preconfiguration-Register C
01142  0000
01143  0000            pcrd    = $d504         Preconfiguration-Register D
01144  0000
01145  0000   In die Preconfiguration-Register A-D wird ein Bitmuster geschrieben,
01146  0000   das durch Schreiben eines beliebigen Wertes in das entsprechende Load
01147  0000   Configuration-Register ins Configuration-Register übernommen wird.
01148  0000
01149  0000
```

```
01150  0000  ---- MMU 8722 Modus Configuration-Register ----
01151  0000
01152  0000           mmumcr = $D505          MMU 8722 Modus Configuration-Register
01153  0000
01154  0000  7.......  Flag für 40/80-Zeichen-BS - Taste:
01155  0000
01156  0000  0xxxxxxx  80-Zeichen BS - Taste nicht gedrückt
01157  0000  1xxxxxxx  40-Zeichen BS - Taste gedrückt
01158  0000
01159  0000  .6......  Flag für C 128 /C 64 - Modus:
01160  0000
01161  0000  x0xxxxxx  C 128 - Modus
01162  0000  x1xxxxxx  C 64 - Modus
01163  0000
01164  0000  ..5.....  Flag für Steckmodul GAME (C 64):
01165  0000
01166  0000  xx0xxxxx  GAME belegt
01157  0000  xx1xxxxx  GAME nicht belegt
01168  0000
01169  0000  ..5.....  Flag für Umschaltung Farbspeicher (C 128):
01170  0000
01171  0000  xx0xxxxx  unbekannt
01172  0000  xx1xxxxx  unbekannt
01173  0000
01174  0000  ...4....  Flag für Steckmodul EXROMIN (C 64):
01175  0000
01176  0000  xxx0xxxx  EXROMIN belegt
01177  0000  xxx1xxxx  EXROMIN nicht belegt
01178  0000
01179  0000  ....3...  FSDIR-Control Bit (FAST SERIAL DEVICE)
01180  0000  xxxx0xxx  FSD Input
01181  0000  xxxx1xxx  FSD Output
01182  0000
01183  0000  .....21.  noch nicht verwendet
01184  0000
01185  0000  .......0  Auswahl des aktiven Prozessors:
01186  0000
01187  0000  xxxxxxx0  Z 80
01188  0000  xxxxxxx1  8502
01180  0000
01190  0000
01191  0000  ----RAM Configuration-Register ----
01192  0000
01193  0000           mmurcr = $d506          RAM Configuration-Register
01194  0000
01195  0000  76......  Auswahl RAM-Bank für VIC-Chip
01196  0000
01197  0000  00xxxxxx  RAM Bank 0
01198  0000  01xxxxxx  RAM Bank 1
01199  0000  10xxxxxx  RAM Bank 2 (wenn nicht vorhanden: Bank 0)
01200  0000  11xxxxxx  RAM Bank 3 (wenn nicht vorhanden: Bank 1)
01201  0000
01202  0000  ..54....  Auswahl 256K-Block (spätere 1MB-Version):
01203  0000
01204  0000  xx00xxxx  256K-Block 0
01205  0000  xx01xxxx  256K-Block 1
01206  0000  xx10xxxx  256K-Block 2
01207  0000  xx11xxxx  256K-Block 3
```

```
01208  0000
01209  0000   ....32..  Definition COMMON RAM AREA:
01210  0000
01211  0000   xxxx00xx  keine  COMMON AREA, unabhängig von Bits1/0
01212  0000   xxxx01xx  untere COMMON AREA
01213  0000   xx10xxxx  obere  COMMON AREA
01214  0000   xx11xxxx  untere und obere COMMON AREA
01215  0000
01216  0000   ......10  Definition Größe der COMMON RAM AREA:
01217  0000
01218  0000   xxxxxx00   1 K
01219  0000   xxxxxx01   4 K
01220  0000   xxxxxx10   8 K
01221  0000   xxxxxx11  16 K
01222  0000
01223  0000
01224  0000  ---- Zeropage Zeiger ----
01225  0000
01226  0000             mmup0l = $d507         Page-0-Pointer low
01227  0000             mmup0h = $d508         Page-0-Pointer high
01228  0000
01229  0000  ---- Stapel Zeiger ----
01230  0000
01231  0000             mmup1l = $d509         Page-1-Pointer low
01232  0000             mmup1h = $d50a         Page-1-Pointer high
01233  0000
01234  0000  ---- Versions Register ----
01235  0000
01236  0000             mmuver = $d50b         Versions-Register
01237  0000
01238  0000   7654....  Bank-Version
01239  0000
01240  0000   0010xxxx   2 64K Banks (kommentierter Rechner)
01241  0000   bis
01242  0000   1111xxxx  15 64K Banks (zukünftige Versionen)
01243  0000
01244  0000   ....3210  MMU - Version
01245  0000
01246  0000   xxxx0000  der kommentierte Rechner besitzt MMU V 0
01247  0000
01248  0000
01249  0000                    .end
01250  0000                    .lib basic.vdc
01251  0000  put"@basic.vdc"
```

```
01252  0000
01253  0000
01254  0000
01255  0000  ===> VDC 8563 Video Display Controller <===
01256  0000
01257  0000  Der Video-Controller VDC 8563 adressiert 16 KByte eigenes RAM für
01258  0000  Zeichengenerator und Attribute.
01259  0000
01260  0000            vdcscn = $0000          Basisadresse VDC BS-RAM
01261  0000            vdcchr = $2000          Basisadresse VDC Zeichengenerator
01262  0000
01263  0000            vdcadr = $d600          VDC 8563 Adreßregister
01264  0000
01265  0000  WRITE: ..543210  Nummer gewähltes Register (0-36)
01266  0000
01267  0000  READ : 7.......  Statusbit des VDC
01268  0000         0xxxxxxx  letzter VDC-Job noch in Arbeit
01269  0000         1xxxxxxx  VDC bereit für nächsten Job
01270  0000
01271  0000  Da die Firma Commodore mit Informationen über von ihr entwickelte
01272  0000  Chips äußerst sparsam umgeht, sind über die weiteren Statusbits
01273  0000  nur Spekulationen möglich (Vergleich mit 6845):
01274  0000
01275  0000         .6......  Status Light-Pen Register:
01276  0000         x0xxxxxx  nach Lesen Reg. 16/17 (LPen)
01277  0000         x1xxxxxx  Light-Pen Strobe festgestellt
01278  0000
01279  0000         ..5.....  Vertical Blanking: Kathodenstrahl
01280  0000         xx0xxxxx  nicht im Rücklauf
01281  0000         xx1xxxxx  im Rücklauf
01282  0000
01283  0000  Die Bits 4-0 werden im 6845 nicht benutzt. Im C 128 werden Bits3-0
01284  0000  des 8563 abgefragt ($e17e). Die Bedeutung ist jedoch unklar.
01285  0000
01286  0000            vdcdat = $d601          VDC 8563 Datenregister
01287  0000
01288  0000  Der Video-Controller VDC 8563 belegt nur die Adressen $d600 und $d601,
01289  0000  verwaltet jedoch intern 36 Register.
01290  0000
01291  0000  Auf die Register wird zum Schreiben/Lesen zugegriffen, indem die ge-
01292  0000  wünschte Registernummer ins Adreßregister geschrieben und das Daten-
01293  0000  register geschrieben/gelesen wird.
01294  0000
01295  0000  Die einzelnen Register haben folgende Bedeutung:
01296  0000
01297  0000  Reg.    Bits
01298  0000  ============
01299  0000
01300  0000    0   76543210  gesamte Anzahl der Zeichen zwischen aufeinander-
01301  0000                  folgenden horizontalen Synchronimplusen-1
01302  0000
01303  0000    1   76543210  Anzahl Zeichen pro Bildschirmzeile (BS-Zeile)
01304  0000
01305  0000    2   76543210  Position des horizontalen Synchronimpulses
01306  0000
01307  0000    3   7654....  Breite des vertikalen Synchronimpulses
01308  0000        ....3210  Breite des horizontalen Synchronimpulses
01309  0000                   (Zeichengenerator-Taktzyklen+1)
```

```
01310  0000
01311  0000    4   76543210  Gesamtzahl BS-Zeilen-1
01312  0000
01313  0000    5   ...43210  Feinabgleich (Anzahl der Bildzeilen)
01314  0000
01315  0000    6   76543210  Anzahl der dargestellten Zeilen
01316  0000
01317  0000    7   76543210  Position des vertikalen Synchronimpulses
01318  0000                  (Anzahl Zeichenzeilen)
01319  0000
01320  0000    8   ......10  Anzeigeform:
01321  0000        xxxxxxx0  kein Zeilensprung
01322  0000        xxxxxx01  Zeilensprung
01323  0000        xxxxxx11  Zeilensprung doppelte Dichte
01324  0000
01325  0000    9   ...43210  Anzahl der Bildzeilen-1 pro BS-Zeile
01326  0000
01327  0000   10   .65.....  Cursor-Modus:
01328  0000        x00xxxxx  Cursor blinkt nicht
01329  0000        x01xxxxx  Cursor ausgeschaltet
01330  0000        x10xxxxx  Cursor blinkt schnell (1/16 Bildwiederholfrequenz)
01331  0000        x11xxxxx  Cursor blinkt langsam (1/32 Bildwiederholfrequenz)
01332  0000
01333  0000        ...43210  Cursor-Anfangszeile im Zeichen (0-7)
01334  0000   11   ...43210  Cursor-Endzeile im Zeichen (0-7)
01335  0000
01336  0000   12   76543210  addrh Startadresse BS-Speicher (Video-RAM)
01337  0000   13   76543210  addrl Startadresse BS-Speicher (Video-RAM)
01338  0000
01339  0000   14   76543210  addrh. akt. Cursorposition im BS-Speicher
01340  0000   15   76543210  addrl. akt. Cursorposition im BS-Speicher
01341  0000
01342  0000   16   76543210  vertikale Lichtgriffel-Pos. (in Bildzeilen+1)
01343  0000   17   76543210  horizontale Lichtgriffel-Pos. (in Bildspalten+1)
01344  0000
01345  0000   18   76543210  addrh akt. Schreib-/Leseposition im BS-Speicher
01346  0000   19   76543210  addrl akt. Schreib-/Leseposition im BS-Speicher
01347  0000
01348  0000   20   76543210  addrh Startadresse Attribut-Speicher
01349  0000   21   76543210  addrl Startadresse Attribut-Speicher
01350  0000
01351  0000   22   7654....  Zeichenbreite in Pixel-1 (einschl. Zwischenraum)
01352  0000        ....3210  Zeichenbreite in Pixel-1 (ohne Zwischenraum)
01353  0000
01354  0000   23   ...43210  Anzahl Bildzeilen/BS-Zeile (ohne Zwischenraum)
01355  0000
01356  0000   24   7.......  Block Copy
01357  0000        0xxxxxxx  Block wird geschr. nach vorausgeg. Schreiben Reg. 30
01358  0000        1xxxxxxx  Block wird kopiert nach vorausgeg. Schreiben Reg. 30
01359  0000
01360  0000        .6......
01361  0000        x0xxxxxx  Normaldarstellung
01362  0000        x1000000  REVERS-Modus
01363  0000
01364  0000        ..5.....  Blinkfrequenz der Bildschirmzeichen
01365  0000        xx0xxxxx  schnell (1/16 Bildwiederholfrequenz)
01366  0000        xx1xxxxx  langsam (1/32 Bildwiederholfrenquez)
01367  0000
```

```
01368  0000          ...43210  Anzahl der BS-Zeilen, die BS nach oben kopiert wird
01369  0000
01370  0000    25    7.......  Betriebsart
01371  0000          0xxxxxxx  Text-Modus
01372  0000          1xxxxxxx  Bit-Map-Modus
01373  0000
01374  0000          .6......  Attribute
01375  0000          x0xxxxxx  kein Attribut, Vordergrundfarbe lt. Reg. 26
01376  0000          x1xxxxxx  Vordergrundfarbe und Attribut lt. Reg. 26
01377  0000
01378  0000          ..5.....  Semigraphik-Modus
01379  0000          xx0xxxxx  horiz. Zwischenraum zweier Zeichen in Hintergrundfarbe
01380  0000          xx1xxxxx  horiz. Zwischenraum wie letztes normal dargest. Zeichen
01381  0000
01382  0000          ...4....  Pixelgröße
01383  0000          xxx0xxxx  = 1 Dot-Takt
01384  0000          xxx1xxxx  = 2 Dot-Takte
01385  0000
01386  0000          ....3210  Anzahl Pixel, um die BS nach links geschoben wird
01387  0000
01388  0000    26    7654....  Vordergrundfarbe, wenn Bit6 von Reg. 25 = 0
01389  0000          ....3210  Hintergrundfarbe für alle Zeichen und Rahmenfarbe
01390  0000
01391  0000    27    76543210  Anzahl Zeichen/BS-Zeile
01392  0000
01393  0000    28    765.....  Adreßbits 15-13 des Zeichengenerators
01394  0000
01395  0000          ...4....  DRAM-Type
01396  0000          xxx0xxxx  4416
01397  0000          xxx1xxxx  4164
01398  0000
01399  0000    29    ...43210  BS-Zeilen Zähler für Unterstreichen
01400  0000
01401  0000    30    76543210  Anzahl Block-Write- bzw. Block-Copy-Zyklen
01402  0000
01403  0000    31    76543210  Daten für Block-Write bzw. normales Write
01404  0000
01405  0000    32    76543210  addrh Quelladresse Block-Copy
01406  0000    33    76543210  addrl Quelladresse Block-Copy
01407  0000
01408  0000    34    76543210  Zeichen von Anfang akt. BS-Zeile bis pos. Flanke,
01409  0000    35    76543210  Zeichen von Anfang akt. BS-Zeile bis neg. Flanke
01410  0000                    des Display-Enable-Pins
01411  0000
01412  0000    36    ....3210  Anzahl der Refresh-Zyklen pro BS-Zeile
01413  0000
01414  0000
01415  0000                      .end
01416  0000                      .lib basic.cia
01417  0000  put"@basic.cia"
```

```
01418  0000
01419  0000
01420  0000  ===> CIA 6526 Complex Interface Adapter  -1- <===
01421  0000
01422  0000            cia1  = $dc00          Basisadresse CIA 6526 -1-
01423  0000
01424  0000  ---- Peripheral Data Register A (PRA) ----
01425  0000
01426  0000            dlpra  = $dc00         Schreib-/Lese-Register Port A
01427  0000
01428  0000   76543210  Spalten Tastaturmatrix (OUTPUTS)
01429  0000
01430  0000  außerdem:
01431  0000   76......  Auswahl Paddle-Set A/B
01432  0000   ...43210  Joystick 0
01433  0000
01434  0000  ---- Peripheral Data Register B (PRB) ----
01435  0000
01436  0000            rows   = $dc01          Schreib-/Leseregister Port B
01437  0000
01438  0000   76543210  Reihen Tastaturmatrix
01439  0000
01440  0000  außerdem:
01441  0000   ...43210  Joystick1
01442  0000
01443  0000  ---- Data Direction Register A (DDRA) ----
01444  0000
01445  0000            diddra = $dc02         CIA1 Datenrichtungs-Reg. a
01446  0000
01447  0000   11111111  alle Bits als Ausgänge geschaltet
01448  0000             zum Scannen der Tastatur
01449  0000
01450  0000  ---- Data Direction Register B (DDRB) ----
01541  0000
01452  0000            diddrb = $dc03         CIA1 Datenrichtungs-Reg. b
01453  0000
01454  0000   00000000  alle Bits als Eingänge geschaltet
01455  0000             zum Scannen der Tastatur
01456  0000
01457  0000  ---- Timer A Low / High Register (TALO/TAHI) ----
01458  0000
01459  0000            d1t1l  = $dc04         benutzt für Serial Device
01460  0000            d1t1h  = $dc05         und Kassettenrecorder
01461  0000
01462  0000  ---- Timer B Low / High Register (TBLO/TBHI) ----
01463  0000
01464  0000            d1t2l  = $dc06         CIA1 TIMER B low
01465  0000            d1t2h  = $dc07         CIA1 TIMER B high
01466  0000
01467  0000  ---- Time of Day (TOD) Zehntelsekunden ----
01468  0000
01469  0000            tod10  = $dc08
01470  0000
01471  0000   7654....  unbenutzt
01472  0000   ....3210  Zehntelsekunden (BCD)
01473  0000
```

```
01474  0000  ---- Time of Day (TOD) Sekunden ----
01475  0000
01476  0000            todsec = $dc09
01477  0000
01478  0000   7654....  Sekunden high (BCD)
01479  0000   ....3210  Sekunden low (BCD)
01480  0000
01481  0000  ---- Time of Day (TOD) Minuten ----
01482  0000
01483  0000            todmin = $dc0a
01484  0000
01485  0000   7654....  Minuten high (BCD)
01486  0000   ....3210  Minuten low (BCD)
01487  0000
01488  0000  ---- Time of Day (TOD) Stunden ----
01489  0000
01490  0000            todhr  = $dc0b
01491  0000
01492  0000   7.......  PM
01493  0000   0xxxxxxx  AM
01494  0000   1xxxxxxx  PM
01495  0000
01496  0000   .65.....  unbenutzt
01497  0000   ...4....  Stunden high
01498  0000   ....3210  Stunden low (BCD)
01499  0000
01500  0000
01501  0000  ---- Serial Data Register (SDR) ----
01502  0000
01503  0000            d1sdr  = $dc0c          CIA1 Serial Data Register
01504  0000
01505  0000   76543210  Serial Data
01506  0000
01507  0000  ---- Interrupt Control Register (ICR) ----
01508  0000
01509  0000            d1icr  = $dc0d          CIA1 Interrupt Control-Register
01510  0000
01511  0000  READ:   7.......  IR IRQ-Flag (1=IRQ)
01512  0000          .65.....  unbenutzt (immer 0)
01513  0000
01514  0000          ...43210  IRQ-Flags:  1=IRQ
01515  0000                                0=NO IRQ
01516  0000          ...4....  FLAG-IRQ (Flag-Input =1)
01517  0000          ....3...  SERIAL PORT (full/empty)
01518  0000          .....2..  ALARM (Alarmzeit erreicht)
01519  0000          ......1.  TIMER B (underflow)
01520  0000          .......0  TIMER A (underflow)
01521  0000
01522  0000  WRITE:  7.......  SET/CLEAR Bit
01523  0000          0xxxxxxx  CLEAR alle Maskenbit =1
01524  0000          1xxxxxxx  SET   alle Maskenbit =1
01525  0000
01526  0000          .65.....  unbenutzt
01527  0000
01528  0000          ...43210  Maskenbits: 1=SET/CLEAR
01529  0000                                0=UNAFFECTED
01530  0000          ...4....  Maskenbit FLAG IRQ
01531  0000          ....3...  Maskenbit SERIAL PORT
```

```
01532  0000           .....2..  Maskenbit ALARM
01533  0000           ......1.  Maskenbit TIMER B
01534  0000           .......0  Maskenbit TIMER A
01535  0000
01536  0000  ---- Control Register A (CRA) ----
01537  0000
01538  0000              d1cra  = $dc0e          CIA1 Control-Register a
01539  0000
01540  0000   7.......  TODIN
01541  0000   0xxxxxxx  60 Hz an TOD-Pin
01542  0000   1xxxxxxx  50 Hz an TOD-Pin
01543  0000
01544  0000   .6......  SPMODE
01545  0000   x0xxxxxx  SERIAL PORT INPUT (ext. Shift-Takt erforderl.)
01546  0000   x1xxxxxx  SERIAL PORT OUTPUT (CNT als Shift-Takt)
01547  0000
01548  0000   ..5.....  INMODE
01549  0000   xx0xxxxx  TIMER A zählt PHI2 Pulse
01550  0000   xx1xxxxx  TIMER A zählt positive CNT-Übergänge
01551  0000
01552  0000   ...4....  LOAD
01553  0000   xxx0xxxx  wirkungslos
01554  0000   xxx1xxxx  FORCE LOAD
01555  0000
01556  0000   ....3...  RUNMODE
01557  0000   xxxx0xxx  CONTINOUS
01558  0000   xxxx1xxx  ONE-SHOT
01559  0000
01560  0000   .....2..  OUTMODE
01561  0000   xxxxx0xx  PULSE
01562  0000   xxxxx1xx  TOGGLE
01563  0000
01564  0000   ......1.  PBON Steuerung PB6
01565  0000   xxxxxx0x  PB6 (Bit6 Port B) normal Operation
01566  0000   xxxxxx1x  PB6 ist TIMER A output
01567  0000
01568  0000   .......0  START TIMER A
01569  0000   xxxxxxx0  Stop TIMER A
01570  0000   xxxxxxx1  Start TIMER A
01571  0000
01572  0000
01573  0000  ---- Control Register B (CRB) ----
01574  0000
01575  0000              d1crb  = $dc0f          CIA1 Control-Register b
01576  0000
01577  0000   7.......  ALARM
01578  0000   0xxxxxxx  Write TOD-Register setzt Uhr
01579  0000   1xxxxxxx  Write TOD-Register setzt Alarm
01580  0000
01581  0000   .65.....  INMODE
01582  0000   x00xxxxx  TIMER B zählt PHI2-Pulse
01583  0000   x01xxxxx  TIMER B zählt pos. CNT-Übergänge
01584  0000   x10xxxxx  TIMER B zählt TIMER A underflow-Pulse
01585  0000   x11xxxxx  TIMER B zählt TIMER A underflow-Pulse wenn CNT high
01586  0000
01587  0000   ...4....  LOAD
01588  0000   xxx0xxxx  wirkungslos
01589  0000   xxx1xxxx  FORCE LOAD
```

```
01590  0000
01591  0000   ....3...  RUNMODE
01592  0000   xxxx0xxx  CONTINOUS
01593  0000   xxxx1xxx  ONE-SHOT
01594  0000
01595  0000   .....2..  OUTMODE
01596  0000   xxxxx0xx  PULSE
01597  0000   xxxxx1xx  TOGGLE
01598  0000
01599  0000   ......1.  PBON Steuerung PB7
01600  0000   xxxxxx0x  PB7 (Bit7 Port B) normal Operation
01601  0000   xxxxxx1x  PB7 ist TIMER B output01602  0000
01603  0000   .......0  START TIMER B
01604  0000   xxxxxxx0  Stop TIMER B
01605  0000   xxxxxxx1  Start TIMER B
01606  0000
01607  0000
01608  0000                    .end
01609  0000                    .lib basic.dma
01610  0000  put"@basic.dma"
```

```
01612  0000
01613  0000
01614  0000  ===> Definitionen DMA Controller 8726 <===
01615  0000
01616  0000
01617  0000   Bis zur Drucklegung gelangte erst eine Platine Commodore 1750 mit
01618  0000   512 KB RAM und dem DMA Controller 8726 zur deutschen Commodore-
01619  0000   Niederlassung in Frankfurt.
01620  0000
01621  0000   Der DMA 8726 ist eine 64beinige Commodore-Entwicklung (ohne Daten-
01622  0000   blatt) und ist neben den 16 RAMs 256 KB x 1 der einzige Chip auf der
01623  0000   RAM EXTENSION 1750.
01624  0000
01625  0000   Die 512 KB RAM Extension soll auch für den C 64 erscheinen.
01626  0000
01627  0000            dma     = $df00          Basisadresse DMA 8726 DMA Controller
01628  0000
01629  0000
01630  0000                    .end
01631  0000                    .lib basic.init
01632  0000  put"@basic.init"
```

```
01634  0000
01635  0000
01636  0000              *       = $4000
01637  4000
01638  4000
01639  4000  ===> BASIC Vektoren <===
01640  4000
01641  4000  4c 23 40    coldvc  jmp hares       Kaltstart - Vektor (BASIC)
01642  4003
01643  4003
01644  4003  4c 09 40    warmvc  jmp sftres      Warmstart - Vektor (BASIC)
01645  4006
01646  4006
01647  4006  4c 4d a8    birqvc  jmp basirq      Sprung über BASIC IRQ-Vector
01648  4009
01649  4009
01650  4009  ===> BASIC Warmstart <===
01651  4009
01652  4009  20 cc ff    sftres  jsr kclrch      ind. I/O-Kanäle schließen (Sprungv.)
01653  400c  20 7a 41            jsr inimmu      MMU Preconfig.-Reg. initialisieren
01654  400f  20 8d 41            jsr stpspr      Sprites stoppen (Geschw. =0)
01655  4012  20 12 41            jsr inivoi      SID-Parameter initialisieren
01656  4015  20 38 52            jsr stkini      reset Stapel- u. BASIC-Befehl-Zeiger
01657  4018
01658  4018  a9 00              lda #_keybd     Kanalnummer Tastatur
01659  401a  85 15              sta channl      aktiver I/O-Kanal
01660  401c
01661  401c
01662  401c  ===> IRQ freigeben, Ausgabe ready <===
01663  401c
01664  401c  58          gordy  cli
01665  401d  4c 37 4d            jmp ready       Ausgabe »ready.« und ready-Status
01666  4020
01667  4020
01668  4020  00 ff ff           .byte $00, $ff, $ff
01669  4023
01670  4023
01671  4023  ===> BASIC Kaltstart <===
01672  4023
01673  4023  20 7a 41    hares   jsr inimmu      MMU Preconfig.-Reg. initialisieren
01674  4026  20 51 42            jsr inivec      BASIC-Vektoren ins RAM kopieren
01675  4029  20 45 40            jsr inisto      BASIC-Parameter initialisieren
01676  402c  20 9b 41            jsr signon      Ausgabe Systemmeldung nach Kaltstart
01677  402f
01678  402f  ad 04 0a            lda initst      Flag BASIC-IRQ: bit0 = 1 (OP-Syst.)
01679  4032  09 01              ora #%00000001  enable BASIC-IRQ (Graphik/Sound)
01680  4034  8d 04 0a            sta initst      Flag BASIC-IRQ: bit0 = 1 (OP-Syst.)
01681  4037
01682  4037  a2 03              ldx #<warmvc    Warmstart - Vektor (BASIC)
01683  4039  8e 00 0a            stx rstvec      Restart-Vektor
01684  403c
01685  403c  a2 fb              ldx #$ff-_hedln Platz für PGM-Zeilen Kopf vor
                                                Eingabepuffer
01686  403e  9a                 txs
01687  403f  20 56 ff           jsr kphenx      Aufruf Kaltstartroutinen
                                                Funktionskarte (Sprungv.)
01688  4042  4c 1c 40           jmp gordy       IRQ freigeben, Ausgabe ready
01689  4045
```

```
01690  4045
01691  4045  ===> BASIC-Parameter initialisieren <===
01692  4045
01693  4045  a9 4c      inisto lda #$4c          JMP-Opcode 6502
01694  4047  85 56             sta jmper        vor Vektor Funktions-Routinen
01695  4049  8d 18 12          sta usrpok       und USR-Routine
01696  404c
01697  404c  a9 28             lda #<fcerr      USR-Vektor mit Adresse
01698  404e  a0 7d             ldy #>fcerr      Behandlung »?illegal quantity error«
01699  4050
01700  4050  8d 19 12          sta usrpok+1     vorbesetzen
01701  4053  8c 1a 12          sty usrpok+2
01702  4056
01703  4056  a9 9f             lda #<flpint     Wandlung GK/INT
01704  4058  a0 84             ldy #>flpint     Wandlung GK/INT
01705  405a
01706  405a  8d 7a 11          sta adray1       ADDRL Vektor Wandlung Gleitk./Int.
01707  405d  8c 7b 11          sty adray1+1     ADDRH Vektor Wandlung Gleitk./Int.
01708  4060
01709  4060  a9 3c             lda #<givayv     16 Bit INT yr/ac nach GK in FAC
01710  4062  a0 79             ldy #>givayv     16 Bit INT yr/ac nach GK in FAC
01711  4064
01712  4064  8d 7c 11          sta adray2       ADDRL Vektor Wandlung Int./Gleitk.
01713  4067  8c 7d 11          sty adray2+1     ADDRH Vektor Wandlung Int./Gleitk.
01714  406a
01715  406a  ===> CHRGET-, ind. Routinen ins COMMON-RAM kopieren <===
01716  406a
01717  406a  a2 55             ldx #inddef-chgtpt
01718  406c  bd 78 42   inis10 lda chgtpt-1,x   Original der chrget-Routine
01719  406f  9d 7f 03          sta chrget-1,x   Kopie im COMMON RAM
01720  4072  ca                dex
01721  4073  d0 f7             bne inis10
01722  4075
01723  4075  ===> versch. Flags mit NULL vorbesetzen <===
01724  4075
01725  4075  8e df 03          stx bits         FAC Overflow Bit
01726  4078  86 15             stx channl       Kanalnummer Tastatur
01727  407a  86 1a             stx lastpt+1     ADDRH letzt benutzte String-Adresse
01728  407c  8e 6f 11          stx trcflg       TRACE-Flag (0=aus, ff=ein)
01729  407f  8e 00 1c          stx rambot       Untergrenze PGM RAM
01730  4082  86 76             stx mvdflg       Flag GRAPHIC-RAM
01731  4084  86 74             stx autinc       Schrittweite low für AUTO
01732  4086  86 75             stx autinc+1     Schrittweite high für AUTO
01733  4088  8e 6b 11          stx width        Strichstärke (0-1)
01734  408b  8e 6a 11          stx scalem       SCALE Flag
01735  408e  8e 6c 11          stx filflg       Flag 1=Ausmalen, 0=nicht
01736  4091  8e 1b 12          stx rndx         GK Variable RND
01737  4094  8e 1c 01          stx dosfa        Primäradresse
01738  4097  8e 76 12          stx tripfg       Flag Bearbeitung Sprite/Sprite Koll.
01739  409a  8e 77 12          stx sbcflg       Flag Bearbeitung Sprite/Hintergr.
                                                Koll.
01740  409d  8e 78 12          stx lpnflg       Flag Bearbeitung Light-Pen
01741  40a0  8e 7f 12          stx intval       Flag für Bearbeitung COLLISION
01742  40a3
01743  40a3  ===> Speicherbereich Sprite-Parameter löschen <===
01744  40a3
01745  40a3  a0 58             ldy #88
01746  40a5  99 7e 11   inis20 sta sspeed,y     Konstante Schritte / BASIC-IRQ
```

```
01747  40a8  88              dey
01748  40a9  10 fa           bpl inis20
01749  40ab
01750  40ab  ===> Zeilen-Link mit lnkdmy ($0101) vorbelegen <===
01751  40ab
01752  40ab  e8              inx
01753  40ac  8e fd 01        stx hilink      High-Byte Vorbelegung Zeilen-Link
01754  40af  8e fc 01        stx lolink      Low-Byte Vorbelegung Zeilen-Link
01755  40b2
01756  40b2  ===> akt. Bank mit System-Bank vorbelegen <===
01757  40b2
01758  40b2  a2 0f           ldx #_sysbk     SYS ROM
01759  40b4  8e d5 03        stx curbnk      akt. Bank
01760  40b7
01761  40b7  ===> Vordergrundfarbe vorbelegen <===
01762  40b7
01763  40b7  a2 0d           ldx #13         hellgrün
01764  40b9  86 86           stx forgnd      Farbcode Vordergrund
01765  40bb
01766  40bb  ===> Multi-Color 1 vorbelegen <===
01767  40bb
01768  40bb  a2 01           ldx #1          weiß
01769  40bd  86 84           stx mulco1      Multi-Color Farbe 1
01770  40bf
01771  40bf  ===> Multi-Color 2 vorbelegen <===
01772  40bf
01773  40bf  a2 02           ldx #2          rot
01774  40c1  86 85           stx mulco2      Zusatzfarbe 2
01775  40c3
01776  40c3  20 5c 6a        jsr setpcl      Vordergr.fb. zu Hintergr./Zusatzf. 1
01777  40c6
01778  40c6  ===> Zeiger in String-Stapel einrichten <===
01779  40c6
01780  40c6  a2 1b           ldx #<tempst    Ende Stapel
01781  40c8  86 18           stx temppt      Zeiger in den String-Stapel
01782  40ca
01783  40ca  ===> Untergrenzen Programm- und Variablenspeicher setzen <===
01784  40ca
01785  40ca  a2 01           ldx #<baswrk    Untergrenze PGM
01786  40cc  a0 1c           ldy #>baswrk    Untergrenze PGM
01787  40ce
01788  40ce  86 2d           stx txttab      ADDRL Untergrenze akt. PGM
01789  40d0  84 2e           sty txttab+1    ADDRH Untergrenze akt. PGM
01790  40d2
01791  40d2  a9 00           lda #<comtop    Obergrenze COMMON RAM
01792  40d4  a0 04           ldy #>comtop    Obergrenze COMMON RAM
01793  40d6
01794  40d6  85 2f           sta vartab      ADDRL Zeiger Anfang einf. Variable
01795  40d8  84 30           sty vartab+1    ADDRH Zeiger Anfang einf. Variable
01796  40da
01797  40da  ===> Obergrenzen Programm- und Variablenspeicher setzen <===
01798  40da
01799  40da  a9 00           lda #<mmucr     MMU 8722 Config. Reg. (Kopie)
01800  40dc  a0 ff           ldy #>mmucr     MMU 8722 Config. Reg. (Kopie)
01801  40de
01802  40de  8d 12 12        sta mxmem0      ADDRL Obergr. PGM RAM
01803  40e1  8c 13 12        sty mxmem0+1    ADDRH Obergr. PGM RAM
01804  40e4
```

```
01805  40e4  a9 00            lda #<mmucr      MMU 8722 Config. Reg. (Kopie)
01806  40e6  a0 ff            ldy #>mmucr      MMU 8722 Config. Reg. (Kopie)
01807  40e8
01808  40e8  85 39            sta mxmem1       ADDRL Obergrenze VAR RAM
01809  40ea  84 3a            sty mxmem1+1     ADDRH Obergrenze VAR RAM
01810  40ec
01811  40ec  ===> Sprite-Zeiger einrichten <===
01812  40ec
01813  40ec  a2 3f            ldx #63
01814  40ee  a0 07            ldy #7
01815  40f0  8a        inis30 txa
01816  40f1  99 f8 07         sta stktop-8,y
01817  40f4  ca               dex
01818  40f5  88               dey
01819  40f6  10 f8            bpl inis30
01820  40f8
01821  40f8  ===> Sprite-Parameter löschen <===
01822  40f8
01823  40f8  a9 00            lda #0
01824  40fa  a2 6c            ldx #108
01825  40fc  9d 7e 11   inis40 sta sspeed,x     Konstante Schritte / BASIC-IRQ
01826  40ff  ca               dex
01827  4100  10 fa            bpl inis40
01828  4102
01829  4102  20 12 41         jsr inivoi       SID-Parameter initialisieren
01830  4105
01831  4105  ===> Basisadressen BS-Zeichengeneratoren eintragen <===
01832  4105
01833  4105  a9 d0            lda #>chrgug     Basisadr. Char.gen. Groß / Graphik
01834  4107  8d ec 11         sta upgrap       ADDRH Char.gen. Groß / Graphik
01835  410a
01836  410a  a9 d8            lda #>chrglu     Basisadresse Char.gen. Klein / Groß
01837  410c  8d eb 11         sta upplow       ADDRH Char.gen. Klein / Groß
01838  410f
01839  410f  4c d9 51         jmp scrtch       BASIC-Zeiger rücksetzen NEW
01840  4112
01841  4112
01842  4112  ===> SID-Parameter initialisieren <===
01843  4112
01844  4112  a9 20      inivoi lda #<_beats/4   Wert für
01845  4114  a0 01            ldy #>_beats/4   Viertelnote
01846  4116
01847  4116  8d 29 12         sta ntime        Low-Byte Tondauer Konst.
01848  4119  8c 2a 12         sty ntime+1      High-Byte Tondauer Konst.
01849  411c
01850  411c  a9 04            lda #4
01851  411e  8d 2b 12         sta octave       OCTAVE Nummer 0-6
01852  4121
01853  4121  a9 10            lda #16
01854  4123  8d 22 12         sta tmprat       Spiel Tempo
01855  4126
01856  4126  a9 00            lda #0
01857  4128  8d 04 d4         sta sid+osc1+oscctl    SID Stimme 1 WELLENFORM
01858  412b  8d 0b d4         sta sid+osc2+oscctl    SID Stimme 2 WELLENFORM
01859  412e  8d 12 d4         sta sid+osc3+oscctl    SID Stimme 3 WELLENFORM
01860  4131  8d fd 12         sta birqfg       BASIC-IRQ Flag (<>0 = in Arbeit)
01861  4134
01862  4134  a9 0f            lda #15
```

```
01863  4136  8d 74 12          sta flters+3     Kopie SID Reg. 24 (VOL/FILT)
01864  4139  8d 75 12          sta oldvol       letzter Wert SID VOL/FILT
01865  413c  8d 18 d4          sta sid+fltvol   SID Filterart / Lautstärke
01866  413f
01867  413f  ===> Tab. 10 Klanghüllkurven ins COMMON RAM <===
01868  413f
01869  413f  a0 1d             ldy #30-1        je 10 atk/dec, sus/rel, osc.ctrl
01870  4141  b9 11 70   iniv10 lda atkmus,y     Original ROM
01871  4144  99 3f 12          sta atktab,y     Kopie COMMON RAM
01872  4147  88                dey
01873  4148  10 f7             bpl iniv10
01874  414a
01875  414a  ===> Tab. 10 Pulsbreiten ins COMMON RAM <===
01876  414a
01877  414a  a2 09             ldx #10-1        10 Werte
01878  414c  bd 2f 70   iniv20 lda pwhmus,x     Tab. PULSBREITE HÜLLK. 0-9
01879  414f  9d 67 12          sta pulshi,x     Tab. High-Byte PULSBREITE akt. HÜLLK.
01880  4152  ca                dex
01881  4153  10 f7             bpl iniv20
01882  4155
01883  4155  ===> Flags setzen für Unterlauf Zähler <===
01884  4155
01885  4155  8e 85 12          stx sndth1       High-Byte Tondauer VOICE 1
01886  4158  8e 86 12          stx sndth2       High-Byte Tondauer VOICE 2
01887  415b  8e 87 12          stx sndth3       High-Byte Tondauer VOICE 3
01888  415e  8e 24 12          stx voichi       Tab. High-Bytes Tondauer (1-3)
01889  4161  8e 26 12          stx voichi+2
01890  4164  8e 28 12          stx voichi+4
01891  4167
01892  4167  ===> Tongeneratoren (Stimmen) 1-3 programmieren <===
01893  4167
01894  4167  a0 02             ldy #3-1         Zähler für 3 Stimmen
01895  4169  8c 2f 12          sty voice        VOICE Nummer -1
01896  416c
01897  416c  a2 00       iniv30 ldx #0
01898  416e  20 b2 6e          jsr stenv1       HÜLLKURVE Nummer ac einstellen
01899  4171
01900  4171  ce 2f 12          dec voice        VOICE Nummer -1
01901  4174  10 f6             bpl iniv30
01902  4176
01903  4176  ee 2f 12   iniv40 inc voice        VOICE Nummer -1
01904  4179  60                rts
01905  417a
01906  417a
01907  417a  ===> MMU Preconfig.-Reg. initialisieren <===
01908  417a
01909  417a  20 45 a8   inimmu jsr mapio        I/O-Chips einblenden
01910  417d
01911  417d  a2 03             ldx #3
01912  417f  bd 89 41   inim10 lda mmutbl,x     Tabelle der MMU Preconfig.-Werte
01913  4182  9d 01 d5          sta pcra,x       MMU 8722 Preconfig. Reg. A
01914  4185  ca                dex
01915  4186  10 f7             bpl inim10
01916  4188
01917  4188  60                rts
01918  4189
01919  4189
```

```
01920  4189  ===> Tabelle der MMU Preconfig.-Werte <===
01921  4189
01922  4189  3f          mmutbl .byte %00111111  Konfiguration »lcra«
01923  418a
01924  418a   00xxxxxx  RAM Bank 0
01925  418a   xx11xxxx  RAM Bank 0       $c000-$cfff
01926  418a   xxxx11xx  RAM Bank 0       $8000-$bfff
01927  418a   xxxxxx11  RAM Bank 0       $d000-$dfff
01928  418a
01929  418a  7f                .byte %01111111  Konfiguration »lcrb«
01930  418b
01931  418b   01xxxxxx  RAM Bank 1
01932  418b   xx11xxxx  RAM Bank 1       $c000-$cfff
01933  418b   xxxx11xx  RAM Bank 1       $8000-$bfff
01934  418b   xxxxxx11  RAM Bank 1       $d000-$dfff
01935  418b
01936  418b  01                .byte %00000001  Konfiguration »lcrc«
01937  418c
01938  418c   00xxxxxx  RAM Bank 0
01939  418c   xx00xxxx  Kernal ROM       $c000-$cfff
01940  418c   xxxx00xx  Teil Basic ROM   $8000-$bfff
01941  418c   xxxxxx0x  Teil BASIC ROM   $4000-$7fff
01942  418c   xxxxxxx1  ROM Char.gen.    $d000-$dfff
01943  418c
01944  418c  41                .byte %01000001  Konfiguration »lcrd«
01945  418d
01946  418d   01xxxxxx  RAM Bank 1
01947  418d   xx00xxxx  Kernal ROM       $c000-$cfff
01948  418d   xxxx00xx  Teil Basic ROM   $8000-$bfff
01949  418d   xxxxxx0x  Teil BASIC ROM   $4000-$7fff
01950  418d   xxxxxxx1  ROM Char.gen.    $d000-$dfff
01951  418d
01952  418d
01953  418d  ===> Sprites stoppen (Geschw. =0) <===
01954  418d
01955  418d  a9 00       stpspr lda #0
01956  418f
01957  418f  a0 07              ldy #7           Zähler für 8 Sprites
01958  4191  be d9 6d   stpsp1 ldx sproff,y     Tabelle Offsets für Parameter
                                                Sprite0-7
01959  4194  9d 7e 11          sta sspeed,x     Konstante Schritte / BASIC-IRQ
01960  4197
01961  4197  88                dey
01962  4198  10 f7             bpl stpsp1
01963  419a  60                rts
01964  419b
01965  419b
01966  419b  ===> Ausgabe Systemmeldung nach Kaltstart <===
01967  419b
01968  419b  a0 00       signon ldy #0
01969  419d  b9 bb 41    sign10 lda sigmsg,y     Systemmeldung nach Kaltstart
01970  41a0  c9 40              cmp #'@'         Startzeichen neue Zeile
01971  41a2  d0 0e              bne sign30
01972  41a4
01973  41a4  24 d7              bit mode         BS-Modus 40/80-Zeichen
01974  41a6  10 0d              bpl sign40
01975  41a8
```

```
01976  41a8  ===> Systemmeldung zentrieren (80 Zeichen BS) <===
01977  41a8
01978  41a8  a2 13            ldx #19          20 Leerzeichen
01979  41aa  a9 20      sign20 lda #_spc
01980  41ac  20 69 92         jsr cbsout       ind. Ausgabe ac auf akt. Kanal
01981  41af  ca               dex
01982  41b0  d0 f8            bne sign20
01983  41b2
01984  41b2  20 69 92   sign30 jsr cbsout       ind. Ausgabe ac auf akt. Kanal
01985  41b5  c8         sign40 iny
01986  41b6
01987  41b6  c0 96            cpy #sigend-sigmsg
01988  41b8  d0 e3            bne sign10
01989  41ba  60               rts
01990  41bb
01991  41bb
01992  41bb  ===> Systemmeldung nach Kaltstart <===
01993  41bb
01994  41bb  93 0d      sigmsg .byte _clr, _cr
01995  41bd  40 20 43         .byte '@ commodore basic v7.0 122365 bytes free', _cr
01995  41c0  4f 4d 4d
01995  41c3  4f 44 4f
01995  41c6  52 45 20
01995  41c9  42 41 53
01995  41cc  49 43 20
01995  41cf  56 37 2e
01995  41d2  30 20 31
01995  41d5  32 32 33
01995  41d8  36 35 20
01995  41db  42 59 54
01995  41de  45 53 20
01995  41e1  46 52 45
01995  41e4  45 0d
01996  41e6  40 20 20         .byte '@   (c)1985 commodore electronics, ltd.', _cr
01996  41e9  20 28 43
01996  41ec  29 31 39
01996  41ef  38 35 20
01996  41f2  43 4f 4d
01996  41f5  4d 4f 44
01996  41f8  4f 52 45
01996  41fb  20 45 4c
01996  41fe  45 43 54
01996  4201  52 4f 4e
01996  4204  49 43 53
01996  4207  2c 20 4c
01996  420a  54 44 2e
01996  420d  0d
01997  420e  40 20 20         .byte '@         (c)1977 microsoft corp.', _cr
01997  4211  20 20 20
01997  4214  20 20 20
01997  4217  20 28 43
01997  421a  29 31 39
01997  421d  37 37 20
01997  4220  4d 49 43
01997  4223  52 4f 53
01997  4226  4f 46 54
01997  4229  20 43 4f
01997  422c  52 50 2e
```

```
01997  422f  0d
01998  4230  40 20 20          .byte '@          all rights reserved', _cr, _eom
01998  4233  20 20 20
01998  4236  20 20 20
01998  4239  20 20 20
01998  423c  41 4c 4c
01998  423f  20 52 49
01998  4242  47 48 54
01998  4245  53 20 52
01998  4248  45 53 45
01998  424b  52 56 45
01998  424e  44 0d 00
01999  4251              sigend
02000  4251
02001  4251
02002  4251  ===> BASIC-Vectoren ins RAM kopieren <===
02003  4251
02004  4251  a2 11       inivec ldx #chgtpt-bvtrs-1
02005  4253  bd 67 42    inivlp lda bvtrs,x      Tabelle der BASIC-Vectoren
02006  4256  9d 00 03           sta ierror,x     ins COMMON-RAM
02007  4259
02008  4259  ca                 dex
02009  425a  10 f7              bpl inivlp
02010  425c
02011  425c  a9 78              lda #<nescfn     »?syntax error« user-def. Funktion
02012  425e  8d fc 02           sta efnvec       Vektor Aufruf user-def. Funktion
                                                 ($4c78)
02013  4261
02014  4261  a9 4c              lda #>nescfn     »?syntax error« user-def. Funktion
02015  4263  8d fd 02           sta efnvec+1     ADDRH Vektor Aufruf user-def
                                                 Funktion
02016  4266
02017  4266  60                 rts
02018  4267
02019  4267
02020  4267  ===> Tabelle der BASIC-Vektoren <===
02021  4267
02022  4267  3f 4d       bvtrs  .word nerror     Behandlung Fehlermeldung
02023  4269  c6 4d              .word nmain      BASIC Eingabeschleife
02024  426b  0d 43              .word ncrnch     Wandlung Text/Tokens
02025  426d  51 51              .word nqplop     LIST/HELP Wandlung Tokens/Text
02026  426f  a2 4a              .word ngone      Test BASIC IRQ, Token auswerten
02027  4271  da 78              .word neval      folg. Ausdruck auswerten
02028  4273  21 43              .word nesclk     Wandlung Text/user-def. Token
02029  4275  cd 51              .word nescpr     Wandlung user-def. Token/Text
02030  4277  a9 4b              .word nescex     Routine zu user-def. Token aufrufen
02031  4279
02032  4279
02033  4279  ===> Original der chrget-Routine <===
02034  4279
02035  4279  e6 3d       chgtpt inc txtptr       ADDRL chrget-Zeiger
02036  427b  d0 02              bne chdgt1
02037  427d  e6 3e              inc txtptr+1     ADDRH chrget-Zeiger
02038  427f
02039  427f  8d 01 ff    chdgt1 sta lcra         64KB PGM-RAM einschalten
02040  4282
02041  4282  a0 00              ldy #0
02042  4284  b1 3d              lda (txtptr),y   ADDRL chrget-Zeiger
```

```
02043  4286
02044  4286  8d 03 ff          sta lcrc          ROM / PGM-RAM einschalten
02045  4289
02046  4289  c9 3a             cmp #':'          Trennzeichen BASIC-Befehle
02047  428b  b0 0a             bcs chdgt2
02048  428d
02049  428d  c9 20             cmp #_spc
02050  428f  f0 e8             beq chgtpt        Spaces überlesen
02051  4291
02052  4291  38                sec
02053  4292  e9 30             sbc #$30
02054  4294
02055  4294  38                sec
02056  4295  e9 d0             sbc #256-$30
02057  4297  60          chdgt2 rts
02058  4298
02059  4298  Beim Verlassen der chrget-Routine ist:
02060  4298
02061  4298  EQUAL-FLAG = 1, wenn Trennzeichen (: oder $00) gelesen wurde,
02062  4298  CARRY-FLAG = 0, wenn Dezimalziffer gelesen wurde
02063  4298
02064  4298
02065  4298  ===> Original iram0: LDA (ac),y PGM-RAM <===
02066  4298
02067  4298  8d a6 03   oiram0 sta iram0+7      LDA (ac),y aus PGM-RAM
02068  429b  8d 01 ff          sta lcra          64KB PGM-RAM einschalten
02069  429e
02070  429e  b1 00             lda (dummy),y     ZP-Zeiger, von iram0 modifiziert
02071  42a0
02072  42a0  8d 03 ff          sta lcrc          ROM / PGM-RAM einschalten
02073  42a3  60                rts
02074  42a4
02075  42a4
02076  42a4  ===> Original iram1: LDA (ac),y VAR-RAM <===
02077  42a4
02078  42a4  8d b2 03   oiram1 sta iram1+7      LDA (ac),y aus VAR-RAM
02079  42a7  8d 02 ff          sta lcrb          64KB VAR-RAM einschalten
02080  42aa
02081  42aa  b1 00             lda (dummy),y     ZP-Zeiger, von iram1 modifiziert
02082  42ac
02083  42ac  8d 04 ff          sta lcrd          ROM / VAR-RAM einschalten
02084  42af  60                rts
02085  42b0
02086  42b0
02087  42b0  ===> Original indin1: LDA (index1),y VAR-RAM <===
02088  42b0
02089  42b0  8d 02 ff   oindi1 sta lcrb          64KB VAR-RAM einschalten
02090  42b3
02091  42b3  b1 24             lda (index1),y
02092  42b5  8d 04 ff          sta lcrd          ROM / VAR-RAM einschalten
02093  42b8  60                rts
02094  42b9
02095  42b9
02096  42b9  ===> Original indin2: LDA (index2),y PGM-RAM <===
02097  42b9
02098  42b9  8d 01 ff   oindi2 sta lcra          64KB PGM-RAM einschalten
02099  42bc
02100  42bc  b1 26             lda (index2),y
```

```
02101  42be  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
02102  42c1  60                rts
02103  42c2
02104  42c2
02105  42c2  ===> Original indtxt: LDA (txtptr),y PGM-RAM <===
02106  42c2
02107  42c2  8d 01 ff   ointxt sta lcra         64KB PGM-RAM einschalten
02108  42c5
02109  42c5  b1 3d              lda (txtptr),y
02110  42c7  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
02111  42ca  60                rts
02112  42cb
02113  42cb
02114  42cb  ===> Kopie zero <===
02115  42cb
02116  42cb  00 00 00   izero  .byte $00, $00, $00
02117  42ce
02118  42ce
02119  42ce  ===> LDA (defpnt),y aus VAR-RAM <===
02120  42ce
02121  42ce  a9 50      inddef lda #<defpnt
02122  42d0  4c ab 03          jmp iram1        LDA (ac),y aus VAR-RAM
02123  42d3
02124  42d3
02125  42d3  ===> LDA (fndpnt),y aus VAR-RAM <===
02126  42d3
02127  42d3  a9 3f      indfrm lda #<fndpnt
02128  42d5  4c ab 03          jmp iram1        LDA (ac),y aus VAR-RAM
02129  42d8
02130  42d8
02131  42d8  ===> LDA (dscpnt),y aus VAR-RAM <===
02132  42d8
02133  42d8  a9 52      inddpt lda #<dscpnt
02134  42da  4c ab 03          jmp iram1        LDA (ac),y aus VAR-RAM
02135  42dd
02136  42dd
02137  42dd  ===> LDA (hightr),y aus PGM-RAM <===
02138  42dd
02139  42dd  a9 5c      inhtr0 lda #<hightr
02140  42df  4c 9f 03          jmp iram0        LDA (ac),y aus PGM-RAM
02141  42e2
02142  42e2
02143  42e2  ===> LDA (hightr),y aus VAR-RAM <===
02144  42e2
02145  42e2  a9 5c      inhtr1 lda #<hightr
02146  42e4  4c ab 03          jmp iram1        LDA (ac),y aus VAR-RAM
02147  42e7
02148  42e7
02149  42e7  ===> LDA (facmo),y aus VAR-RAM <===
02150  42e7
02151  42e7  a9 66      indfmo lda #<facmo
02152  42e9  4c ab 03          jmp iram1        LDA (ac),y aus VAR-RAM
02153  42ec
02154  42ec
02155  42ec  ===> LDA (lowtr),y aus PGM-RAM <===
02156  42ec
02157  42ec  a9 61      indlo0 lda #<lowtr
02158  42ee  4c 9f 03          jmp iram0        LDA (ac),y aus PGM-RAM
```

```
02159  42f1
02160  42f1
02161  42f1  ===> LDA (strng1),y aus PGM-RAM <===
02162  42f1
02163  42f1  a9 70      indst0 lda #<strng1
02164  42f3  4c 9f 03          jmp iram0        LDA (ac),y aus PGM-RAM
02165  42f6
02166  42f6
02167  42f6
02167  42f6  ===> LDA (strng1),y aus VAR-RAM <===
02168  42f6
02169  42f6  a9 70      indst1 lda #<strng1
02170  42f8  4c ab 03          jmp iram1        LDA (ac),y aus VAR-RAM
02171  42fb
02172  42fb
02173  42fb  ===> LDA (grbpnt),y aus VAR-RAM <===
02174  42fb
02175  42fb  a9 50      indgrb lda #<grbpnt
02176  42fd  4c ab 03          jmp iram1        LDA (ac),y aus VAR-RAM
02177  4300
02178  4300
02179  4300  ===> LDA (lowtr),y aus VAR-RAM <===
02180  4300
02181  4300  a9 61      indlo1 lda #<lowtr
02182  4302  4c ab 03          jmp iram1        LDA (ac),y aus VAR-RAM
02183  4305
02184  4305
02185  4305  ===> LDA (index1),y aus PGM-RAM <===
02186  4305
02187  4305  a9 24      ind1_0 lda #<index1
02188  4307  4c 9f 03          jmp iram0        LDA (ac),y aus PGM-RAM
02189  430a
02190  430a
02191  430a                    .end
02192  430a                    .lib basic.crunch
02193  430a  put"@basic.crunch"
```

```
02195  430a
02196  430a
02197  430a  ===> ind. Wandlung Text/Tokens <===
02198  430a
02199  430a  6c 04 03   crunch jmp (icrnch)     ind. Wandlung Text/Tokens ($430d)
02200  430d
02201  430d
02202  430d  ===> Wandlung Text/Tokens <===
02203  430d
02204  430d  a5 3d      ncrnch lda txtptr       ADDRL chrget-Zeiger
02205  430f  48               pha
02206  4310
02207  4310  a5 3e             lda txtptr+1     ADDRH chrget-Zeiger
02208  4312  48               pha
02209  4313
02210  4313
02211  4313  ===> Zeilennr. überlesen, nächst. Befehl <===
02212  4313
02213  4313  20 86 03   crun05 jsr chrget       letztes Zeichen erneut nach ac
02214  4316  4c 1c 43         jmp crun20
02215  4319
02216  4319
02217  4319  ===> Zeilennr. überlesen, nächst. Befehl <===
02218  4319
02219  4319  20 80 03   crun10 jsr chrget       nächstes Zeichen nach ac
02220  431c  90 fb      crun20 bcc crun10       Zeilennr. überlesen, nächst. Befehl
02221  431e
02222  431e  ===> ind. Wandlung Text/user-def. Tokens <===
02223  431e
02224  431e  6c 0c 03         jmp (iesclk)     ind. Wandlg. Text/user-def. Token
                                                ($4321)
02225  4321
02226  4321
02227  4321  ===> Wandlung Text/user-def. Token <===
02228  4321
02229  4321  b0 03      nesclk bcs crun25       kein user-def. Token
02230  4323  4c b2 43         jmp crun95       BASIC-Wort durch (ESC- und)
                                                BASIC-Token ersetzen
02231  4326
02232  4326
02233  4326  ===> Test auf ':', '?' (print), pi, Anführungszeichen <===
02234  4326
02235  4326  c9 00      crun25 cmp #_eom        Zeilenende
02236  4328  f0 77             beq crun90       Token-Umwandlung Ende
02237  432a
02238  432a  c9 3a             cmp #':'         Trennzeichen
02239  432c  f0 eb             beq crun10       Zeilennr. überlesen, nächst. Befehl
02240  432e
02241  432e  c9 3f             cmp #'?'         Abkürzung für PRINT
02242  4330  d0 04             bne crun30
02243  4332
02244  4332  a9 99             lda #_print      PRINT Token
02245  4334  d0 50             bne crun75
02246  4336
02247  4336
02248  4336  c9 80      crun30 cmp #$80         geshiftetes Zeichen
02249  4338  90 0b             bcc crun40       Text in Anführungszeichen überlesen
02250  433a
```

```
02251  433a  c9 ff            cmp #_pi          Code für pi
02252  433c  f0 db            beq crun10        Zeilennr. überlesen, nächst. Befehl
02253  433e
02254  433e  a0 01            ldy #1
02255  4340  20 cc 43         jsr kloop         Zeilenrest hinter Token kopieren
02256  4343  f0 ce            beq crun05        immer
02257  4345
02258  4345
02259  4345  ===> Text in Anführungszeichen überlesen <===
02260  4345
02261  4345  c9 22      crun40 cmp #'"'
02262  4347  d0 0d            bne crnesc        Suchen in ESC COMMAND Tabelle
02263  4349
02264  4349  20 80 03   crun50 jsr chrget       nächstes Zeichen nach ac
02265  434c
02266  434c  c9 00            cmp #$00          Zeilenende
02267  434e  f0 51            beq crun90        Token-Umwandlung Ende
02268  4350
02269  4350  c9 22            cmp #'"'
02270  4352  f0 c5            beq crun10        Zeilennr. überlesen, nächst. Befehl
02271  4354  d0 f3            bne crun50        immer
02272  4356
02273  4356
02274  4356  ===> Suchen in ESC COMMAND Tabelle   <===
02275  4356
02276  4356  a9 46      crnesc lda #>esccmd     BASIC ESC COMMAND Tabelle
02277  4358  a0 09            ldy #<esccmd      BASIC ESC COMMAND Tabelle
02278  435a  20 e2 43         jsr reser         Text in Tabelle ac/yr suchen
02279  435d  90 06            bcc crne10        nicht gefunden
02280  435f
02281  435f  ===> ESC COMMAND- u. BASIC-Token einsetzen <===
02282  435f
02283  435f  a9 81            lda #$81          Offset für BASIC-Token
02284  4361  a2 00            ldx #0            Flag für ESC COMMAND-Token
02285  4363  f0 4b            beq crun94        immer
02286  4365
02287  4365
02288  4365  ===> Suchen in ESC FUNCTION Tabelle   <===
02289  4365
02290  4365  a9 46      crne10 lda #>escfn      BASIC ESC FUNCTION Tabelle
02291  4367  a0 c9            ldy #<escfn       BASIC ESC FUNCTION Tabelle
02292  4369  20 e2 43         jsr reser         Text in Tabelle ac/yr suchen
02293  436c  90 06            bcc crne20        nicht gefunden
02294  436e
02295  436e  ===> ESC FUNCTION- u. BASIC-Token einsetzen <===
02296  436e
02297  436e  a9 81            lda #$81          Offset für BASIC-Token
02298  4370  a2 ff            ldx #$ff          Flag für ESC FUNCTION-Token
02299  4372  d0 3c            bne crun94        immer
02300  4374
02301  4374
02302  4374  ===> Suchen in Befehlswort-Tabelle   <===
02303  4374
02304  4374  a9 44      crne20 lda #>reslst     BASIC Befehlswort Tabelle
02305  4376  a0 17            ldy #<reslst      BASIC Befehlswort Tabelle
02306  4378  20 e2 43         jsr reser         Text in Tabelle ac/yr suchen
02307  437b  90 9c            bcc crun10        nicht gefunden
02308  437d
```

```
02309  437d  c0 00           cpy #0          Zeiger auf Ende BASIC-Befehl
02310  437f  f0 03           beq crne30      Ein-Zeichen-Befehl
02311  4381
02312  4381  20 cc 43        jsr kloop       Zeilenrest hinter Token kopieren
02313  4384  a5 0d     crne30 lda count       Zähler BASIC-Befehle
02314  4386  a0 00     crun75 ldy #0          ac an erster Spalte
02315  4388  91 3d           sta (txtptr),y  Befehlswort speichern
02316  438a
02317  438a  c9 8f           cmp #_remt      REM Token
02318  438c  f0 0d           beq crun80      REM: Zeiger hinter Zeilenende
02319  438e
02320  438e  c9 83           cmp #_datat     DATA Token
02321  4390  d0 87           bne crun10      Zeilennr. überlesen, nächst. Befehl
02322  4392
02323  4392  ===> DATA: Zeiger hinter Datas <===
02324  4392
02325  4392  20 80 03        jsr chrget      nächstes Zeichen nach ac
02326  4395  20 8f 52        jsr data        BASIC-Statement DATA ($83)
02327  4398  4c 13 43        jmp crun05      Zeilennr. überlesen, nächst. Befehl
02328  439b
02329  439b
02330  439b  ===> REM: Zeiger hinter Zeilenende <===
02331  439b
02332  439b  20 80 03  crun80 jsr chrget      nächstes Zeichen nach ac
02333  439e  20 9d 52        jsr rem         BASIC-Statement REM ($8f)
02334  43a1
02335  43a1
02336  43a1  ===> Token-Umwandlung Ende <===
02337  43a1
02338  43a1  a6 3d      crun90 ldx txtptr     ADDRL chrget-Zeiger
02339  43a3
02340  43a3  68               pla
02341  43a4  85 3e            sta txtptr+1    ADDRH chrget-Zeiger
02342  43a6
02343  43a6  68               pla
02344  43a7  85 3d            sta txtptr      ADDRL chrget-Zeiger
02345  43a9
02346  43a9  ===> neue Zeilenlänge nach yr <===
02347  43a9
02348  43a9  38               sec
02349  43aa  8a               txa             Zeiger auf Zeilenende
02350  43ab  e5 3d            sbc txtptr      minus Zeiger auf Zeilenanfang
02351  43ad  a8               tay
02352  43ae  c8               iny             plus 1 = Zeilenlänge
02353  43af  60               rts
02354  43b0
02355  43b0
02356  43b0  ===> Test auf Aufruf user-def. BASIC-Erweiterung <===
02357  43b0
02358  43b0  65 0d      crun94 adc count      Offset + Nummer BASIC-Befehl = Token
02359  43b2
02360  43b2
02361  43b2  ===> BASIC-Wort durch (ESC- und) BASIC-Token ersetzen <===
02362  43b2
02363  43b2  48         crun95 pha            Token
02364  43b3
02365  43b3  88               dey             wegen zwei Tokens
02366  43b4  20 cc 43         jsr kloop       Zeilenrest hinter Token kopieren
```

```
02367  43b7
02368  43b7  a9 fe            lda #_ecmdt      ESC COMMAND Token
02369  43b9
02370  43b9  e8               inx              Flag = $ff für ESC FUNCTION ?
02371  43ba  d0 02            bne crun96
02372  43bc
02373  43bc  a9 ce            lda #_efunt      ESC FUNCTION Token
02374  43be  a0 00     crun96 ldy #0           ac an erster Stelle
02375  43c0  91 3d            sta (txtptr),y   Befehlswort abspeichern
02376  43c2
02377  43c2  c8               iny              an zweiter Stelle
02378  43c3  68               pla              Token
02379  43c4  91 3d            sta (txtptr),y   abspeichern
02380  43c6
02381  43c6  20 80 03         jsr chrget       nächstes Zeichen nach ac
02382  43c9  4c 19 43         jmp crun10       Zeilennr. überlesen, nächst. Befehl
02383  43cc
02384  43cc
02385  43cc  ===> Zeilenrest hinter Token kopieren <===
02386  43cc
02387  43cc  18         kloop  clc
02388  43cd  98               tya              Zeilenlänge
02389  43ce  65 3d            adc txtptr       plus akt.chrget-Zeiger
02390  43d0  85 24            sta index1       = Zeiger hinter BASIC-Wort
02391  43d2
02392  43d2  a5 3e            lda txtptr+1     ADDRH chrget-Zeiger
02393  43d4  69 00            adc #0
02394  43d6  85 25            sta index1+1
02395  43d8
02396  43d8  a0 ff            ldy #$ff
02397  43da  c8         kloop2 iny
02398  43db  b1 24            lda (index1),y   Zeilenrest
02399  43dd  91 3d            sta (txtptr),y   vorkopieren
02400  43df  d0 f9            bne kloop2
02401  43e1  60               rts
02402  43e2
02403  43e2
02404  43e2  ===> Text in Tabelle ac/yr suchen <===
02405  43e2
02406  43e2  85 25      reser  sta index1+1
02407  43e4  84 24            sty index1
02408  43e6
02409  43e6  a0 00            ldy #0           Zähler für BASIC-Wörter
02410  43e8  84 0d            sty count        initialisieren
02411  43ea
02412  43ea  88               dey
02413  43eb
02414  43eb
02415  43eb  ===> nächstes Zeichen vergleichen <===
02416  43eb
02417  43eb  c8         rese10 iny
02418  43ec  b1 3d      rese20 lda (txtptr),y   Zeichen aus BASIC-Wort
02419  43ee  38               sec
02420  43ef  f1 24            sbc (index1),y   minus Zeichen aus Tabelle
02421  43f1  f0 f8            beq rese10       nächstes Zeichen vergleichen
02422  43f3
02423  43f3  c9 80            cmp #$80         Bit7 =1, Wortende
02424  43f5  f0 1b            beq rese60       Token in count berechnen
```

```
02425  43f7
02426  43f7
02427  43f7
02427  43f7  ===> Tabelle bis Wortende überlesen <===
02428  43f7
02429  43f7  b1 24      rese30 lda (index1),y   Zeichen aus Tabelle
02430  43f9  30 03             bmi rese40       Tabellen-Zeiger hinter Wortende
02431  43fb
02432  43fb  c8                iny
02433  43fc  d0 f9             bne rese30       Tabelle bis Wortende überlesen
02434  43fe
02435  43fe
02436  43fe  ===> Tabellen-Zeiger hinter Wortende <===
02437  43fe
02438  43fe  c8         rese40 iny
02439  43ff  e6 0d             inc count
02440  4401
02441  4401  18                clc
02442  4402  98                tya
02443  4403  65 24             adc index1
02444  4405  85 24             sta index1
02445  4407
02446  4407  90 02             bcc rese50       erstes Zeichen aus nächstem
                                                Befehlstext nach ac
02447  4409  e6 25             inc index1+1
02448  440b
02449  440b
02450  440b  ===> erstes Zeichen aus nächstem Befehlstext nach ac <===
02451  440b
02452  440b  18         rese50 clc
02453  440c  a0 00             ldy #0
02454  440e  b1 24             lda (index1),y
02455  4410  d0 da             bne rese20       noch nicht Endezeichen
02456  4412
02457  4412
02458  4412  ===> Token in count berechnen <===
02459  4412
02460  4412  05 0d      rese60 ora count
02461  4414  85 0d             sta count
02462  4416  60                rts
02463  4417
02464  4417
02465  4417                    .end
02466  4417                    .lib basic.reslst
02467  4417  put"@basic.reslst"
```

```
02469  4417
02470  4417
02471  4417  ===> BASIC Befehlswort Tabelle <===
02472  4417
02473  4417  45 4e c4   reslst .byte 'enD'      $80
02474  441a              _endt  = $80
02475  441a  46 4f d2           .byte 'foR'      $81
02476  441d              _fort  = $81
02477  441d  4e 45 58           .byte 'nexT'     $82
02477  4420  d4
02478  4421              _nextt = $82
02479  4421  44 41 54           .byte 'datA'     $83
02479  4424  c1
02480  4425              _datat = $83
02481  4425  49 4e 50           .byte 'input', $a3     $84
02481  4428  55 54 a3
02482  442b  49 4e 50           .byte 'inpuT'    $85
02482  442e  55 d4
02483  4430  44 49 cd           .byte 'diM'      $86
02484  4433  52 45 41           .byte 'reaD'     $87
02484  4436  c4
02485  4437  4c 45 d4           .byte 'leT'      $88
02486  443a  47 4f 54           .byte 'gotO'     $89
02486  443d  cf
02487  443e              _gotot = $89
02488  443e  52 55 ce           .byte 'ruN'      $8a
02489  4441              _runt  = $8a
02490  4441  49 c6              .byte 'iF'       $8b
02491  4443  52 45 53           .byte 'restorE'  $8c
02491  4446  54 4f 52
02491  4449  c5
02492  444a              _restt = $8c
02493  444a  47 4f 53           .byte 'gosuB'    $8d
02493  444d  55 c2
02494  444f              _gosut = $8d
02495  444f  52 45 54           .byte 'returN'   $8e
02495  4452  55 52 ce
02496  4455  52 45 cd           .byte 'reM'      $8f
02497  4458              _remt  = $8f
02498  4458  53 54 4f           .byte 'stoP'     $90
02498  445b  d0
02499  445c  4f ce              .byte 'oN'       $91
02500  445e              _ont   = $91
02501  445e  57 41 49           .byte 'waiT'     $92
02501  4461  d4
02502  4462  4c 4f 41           .byte 'loaD'     $93
02502  4465  c4
02503  4466  53 41 56           .byte 'savE'     $94
02503  4469  c5
02504  446a  56 45 52           .byte 'verifY'   $95
02504  446d  49 46 d9
02505  4470  44 45 c6           .byte 'deF'      $96
02506  4473  50 4f 4b           .byte 'pokE'     $97
02506  4476  c5
02507  4477  50 52 49           .byte 'print', $a3     $98
02507  447a  4e 54 a3
02508  447d  50 52 49           .byte 'prinT'    $99
02508  4480  4e d4
```

```
02509  4482                _print = $99
02510  4482  43 4f 4e             .byte 'conT'      $9a
02510  4485  d4
02511  4486  4c 49 53             .byte 'lisT'      $9b
02511  4489  d4
02512  448a  43 4c d2             .byte 'clR'       $9c
02513  448d                _clrt  = $9c
02514  448d  43 4d c4             .byte 'cmD'       $9d
02515  4490  53 59 d3             .byte 'syS'       $9e
02516  4493  4f 50 45             .byte 'opeN'      $9f
02516  4496  ce
02517  4497  43 4c 4f             .byte 'closE'     $a0
02517  449a  53 c5
02518  449c  47 45 d4             .byte 'geT'       $a1
02519  449f  4e 45 d7             .byte 'neW'       $a2
02520  44a2                _scrat = $a2
02521  44a2  54 41 42             .byte 'tab', $a8 $a3
02521  44a5  a8
02522  44a6                _tabt  = $a3
02523  44a6  54 cf                .byte 'tO'        $a4
02524  44a8                _tot   = $a4
02525  44a8  46 ce                .byte 'fN'        $a5
02526  44aa                _fnt   = $a5
02527  44aa  53 50 43             .byte 'spc', $a8 $a6
02527  44ad  a8
02528  44ae                _spct  = $a6
02529  44ae  54 48 45             .byte 'theN'      $a7
02529  44b1  ce
02530  44b2                _thent = $a7
02531  44b2  4e 4f d4             .byte 'noT'       $a8
02532  44b5                _nott  = $a8
02533  44b5  53 54 45             .byte 'steP'      $a9
02533  44b8  d0
02534  44b9                _stept = $a9
02535  44b9  ab                   .byte $ab         $aa   '+'
02536  44ba                _plust = $aa
02537  44ba  ad                   .byte $ad         $ab   '-'
02538  44bb                _minut = $ab
02539  44bb  aa                   .byte $aa         $ac   '*'
02540  44bc  af                   .byte $af         $ad   '/'
02541  44bd  de                   .byte $de         $ae
02542  44be  41 4e c4             .byte 'anD'       $af
02543  44c1  4f d2                .byte 'oR'        $b0
02544  44c3  be                   .byte $be         $b1   '>'
02545  44c4                _great = $b1
02546  44c4  bd                   .byte $bd         $b2   '='
02547  44c5                _eqult = $b2
02548  44c5  bc                   .byte $bc         $b3   '<'
02549  44c6                _lesst = $b3
02550  44c6  53 47 ce             .byte 'sgN'       $b4
02551  44c9                       _onefn = $b4
02552  44c9  49 4e d4             .byte 'inT'       $b5
02553  44cc  41 42 d3             .byte 'abS'       $b6
02554  44cf  55 53 d2             .byte 'usR'       $b7
02555  44d2  46 52 c5             .byte 'frE'       $b8
02556  44d5  50 4f d3             .byte 'poS'       $b9
02557  44d8  53 51 d2             .byte 'sqR'       $ba
02558  44db  52 4e c4             .byte 'rnD'       $bb
```

```
02559  44de  4c 4f c7          .byte 'loG'       $bc
02560  44e1  45 58 d0          .byte 'exP'       $bd
02561  44e4  43 4f d3          .byte 'coS'       $be
02562  44e7  53 49 ce          .byte 'siN'       $bf
02563  44ea  54 41 ce          .byte 'taN'       $c0
02564  44ed  41 54 ce          .byte 'atN'       $c1
02565  44f0  50 45 45          .byte 'peeK'      $c2
02565  44f3  cb
02566  44f4  4c 45 ce          .byte 'leN'       $c3
02567  44f7  53 54 52          .byte 'str', $a4 $c4
02567  44fa  a4
02568  44fb  56 41 cc          .byte 'vaL'       $c5
02569  44fe  41 53 c3          .byte 'asC'       $c6
02570  4501  43 48 52          .byte 'chr', $a4 $c7
02570  4504  a4
02571  4505  4c 45 46          .byte 'left', $a4      $c8
02571  4508  54 a4
02572  450a                _leftt = $c8
02573  450a  52 49 47          .byte 'right', $a4     $c9
02573  450d  48 54 a4
02574  4510  4d 49 44          .byte 'mid', $a4 $ca
02574  4513  a4
02575  4514                _middt = $ca
02576  4514  47 cf             .byte 'gO'        $cb
02577  4516                _got   = $cb
02578  4516  52 47 d2          .byte 'rgR'       $cc
02579  4519                _rgrt  = $cc
02580  4519  52 43 4c          .byte 'rclR'      $cd
02580  451c  d2
02581  451d  80                .byte $80         $ce (ESC FUNCTION Token)
02582  451e                _efunt = $ce
02583  451e  4a 4f d9          .byte 'joY'       $cf
02584  4521  52 44 4f          .byte 'rdoT'      $d0
02584  4524  d4
02585  4525  44 45 c3          .byte 'deC'       $d1
02586  4528  48 45 58          .byte 'hex', $a4 $d2
02586  452b  a4
02587  452c  45 52 52          .byte 'err', $a4 $d3
02587  452f  a4
02588  4530  49 4e 53          .byte 'instR'     $d4
02588  4533  54 d2
02589  4535                _instt = $d4
02590  4535                _hofnt = $d4              höchster alter Funktions Token
02591  4535  45 4c 53          .byte 'elsE'      $d5
02591  4538  c5
02592  4539                _elset = $d5
02593  4539  52 45 53          .byte 'resumE'    $d6
02593  453c  55 4d c5
02594  453f                _resut = $d6
02595  453f  54 52 41          .byte 'traP'      $d7
02595  4542  d0
02596  4543                _trapt = $d7
02597  4543  54 52 4f          .byte 'troN'      $d8
02597  4546  ce
02598  4547  54 52 4f          .byte 'trofF'     $d9
02598  454a  46 c6
02599  454c  53 4f 55          .byte 'sounD'     $da
02599  454f  4e c4
```

```
02600  4551  56 4f cc          .byte 'voL'       $db
02601  4554  41 55 54          .byte 'autO'      $dc
02601  4557  cf
02602  4558  50 55 44          .byte 'pudeF'     $dd
02602  455b  45 c6
02603  455d  47 52 41          .byte 'graphiC'   $de
02603  4560  50 48 49
02603  4563  c3
02604  4564  50 41 49          .byte 'painT'     $df
02604  4567  4e d4
02605  4569  43 48 41          .byte 'chaR'      $e0
02605  456c  d2
02606  456d  42 4f d8          .byte 'boX'       $e1
02607  4570  43 49 52          .byte 'circlE'    $e2
02607  4573  43 4c c5
02608  4576  47 53 48          .byte 'gshapE'    $e3
02608  4579  41 50 c5
02609  457c  53 53 48          .byte 'sshapE'    $e4
02609  457f  41 50 c5
02610  4582  44 52 41          .byte 'draW'      $e5
02610  4585  d7
02611  4586  4c 4f 43          .byte 'locatE'    $e6
02611  4589  41 54 c5
02612  458c  43 4f 4c          .byte 'coloR'     $e7
02612  458f  4f d2
02613  4591  53 43 4e          .byte 'scnclR'    $e8
02613  4594  43 4c d2
02614  4597  53 43 41          .byte 'scalE'     $e9
02614  459a  4c c5
02615  459c  48 45 4c          .byte 'helP'      $ea
02615  459f  d0
02616  45a0  44 cf             .byte 'dO'        $eb
02617  45a2              _dot   = $eb
02618  45a2  4c 4f 4f          .byte 'looP'      $ec
02618  45a5  d0
02619  45a6              _loopt = $ec
02620  45a6  45 58 49          .byte 'exiT'      $ed
02620  45a9  d4
02621  45aa  44 49 52          .byte 'directorY'       $ee
02621  45ad  45 43 54
02621  45b0  4f 52 d9
02622  45b3  44 53 41          .byte 'dsavE'     $ef
02622  45b6  56 c5
02623  45b8  44 4c 4f          .byte 'dloaD'     $f0
02623  45bb  41 c4
02624  45bd  48 45 41          .byte 'headeR'    $f1
02624  45c0  44 45 d2
02625  45c3  53 43 52          .byte 'scratcH'   $f2
02625  45c6  41 54 43
02625  45c9  c8
02626  45ca  43 4f 4c          .byte 'collecT'   $f3
02626  45cd  4c 45 43
02626  45d0  d4
02627  45d1  43 4f 50          .byte 'copY'      $f4
02627  45d4  d9
02628  45d5  52 45 4e          .byte 'renamE'    $f5
02628  45d8  41 4d c5
02629  45db  42 41 43          .byte 'backuP'    $f6
```

```
02629  45de  4b 55 d0
02630  45e1  44 45 4c          .byte 'deletE'   $f7
02630  45e4  45 54 c5
02631  45e7  52 45 4e          .byte 'renumbeR' $f8
02631  45ea  55 4d 42
02631  45ed  45 d2
02632  45ef  4b 45 d9          .byte 'keY'      $f9
02633  45f2               _keyt  = $f9
02634  45f2  4d 4f 4e          .byte 'monitoR'  $fa
02634  45f5  49 54 4f
02634  45f8  d2
02635  45f9               _monit = $fa
02636  45f9  55 53 49          .byte 'usinG'    $fb
02636  45fc  4e c7
02637  45fe               _usint = $fb
02638  45fe  55 4e 54          .byte 'untiL'    $fc
02638  4601  49 cc
02639  4603               _untlt = $fc
02640  4603  57 48 49          .byte 'whilE'    $fd
02640  4606  4c c5
02641  4608               _whilt = $fd
02642  4608  00                .byte _eom
02643  4609               _ecmdt = $fe
02644  4609
02645  4609
02646  4609  ===> BASIC ESC COMMAND Tabelle   <===
02647  4609
02648  4609  42 41 4e   esccmd .byte 'banK'     $fe $02
02648  460c  cb
02649  460d               _lecmt = $02          niedrigster ESC COMMAND Token
02650  460d  46 49 4c          .byte 'filteR'   $fe $03
02650  4610  54 45 d2
02651  4613  50 4c 41          .byte 'plaY'     $fe $04
02651  4616  d9
02652  4617  54 45 4d          .byte 'tempO'    $fe $05
02652  461a  50 cf
02653  461c  4d 4f 56          .byte 'movspR'   $fe $06
02653  461f  53 50 d2
02654  4622  53 50 52          .byte 'spritE'   $fe $07
02654  4625  49 54 c5
02655  4628  53 50 52          .byte 'sprcoloR' $fe $08
02655  462b  43 4f 4c
02655  462e  4f d2
02656  4630  52 52 45          .byte 'rreG'     $fe $09
02656  4633  c7
02657  4634  45 4e 56          .byte 'envelopE' $fe $0a
02657  4637  45 4c 4f
02657  463a  50 c5
02658  463c  53 4c 45          .byte 'sleeP'    $fe $0b
02658  463f  45 d0
02659  4641  43 41 54          .byte 'cataloG'  $fe $0c
02659  4644  41 4c 4f
02659  4647  c7
02660  4648  44 4f 50          .byte 'dopeN'    $fe $0d
02660  464b  45 ce
02661  464d  41 50 50          .byte 'appenD'   $fe $0e
02661  4650  45 4e c4
02662  4653  44 43 4c          .byte 'dclosE'   $fe $0f
```

```
02662  4656  4f 53 c5
02663  4659  42 53 41          .byte 'bsavE'    $fe $10
02663  465c  56 c5
02664  465e  42 4c 4f          .byte 'bloaD'    $fe $11
02664  4661  41 c4
02665  4663  52 45 43          .byte 'recorD'   $fe $12
02665  4666  4f 52 c4
02666  4669  43 4f 4e          .byte 'concaT'   $fe $13
02666  466c  43 41 d4
02667  466f  44 56 45          .byte 'dverifY'  $fe $14
02667  4672  52 49 46
02667  4675  d9
02668  4676  44 43 4c          .byte 'dcleaR'   $fe $15
02668  4679  45 41 d2
02669  467c  53 50 52          .byte 'sprsaV'   $fe $16
02669  467f  53 41 d6
02670  4682  43 4f 4c          .byte 'collisioN'      $fe $17
02670  4685  4c 49 53
02670  4688  49 4f ce
02671  468b                _colit = $17
02672  468b  42 45 47          .byte 'begiN'    $fe $18
02672  468e  49 ce
02673  4690                _begnt = $18
02674  4690  42 45 4e          .byte 'benD'     $fe $19
02674  4693  c4
02675  4694                _bendt = $19
02676  4694  57 49 4e          .byte 'windoW'   $fe $1a
02676  4697  44 4f d7
02677  469a  42 4f 4f          .byte 'booT'     $fe $1b
02677  469d  d4
02678  469e  57 49 44          .byte 'widtH'    $fe $1c
02678  46a1  54 c8
02679  46a3  53 50 52          .byte 'sprdeF'   $fe $1d
02679  46a6  44 45 c6
02680  46a9  51 55 49          .byte 'quiT'     $fe $1e
02680  46ac  d4
02681  46ad  53 54 41          .byte 'stasH'    $fe $1f
02681  46b0  53 c8
02682  46b2  a0                .byte $a0        $fe $20 = Space
02683  46b3  46 45 54          .byte 'fetcH'    $fe $21
02683  46b6  43 c8
02684  46b8  a0                .byte $a0        $fe $22 = Anführungszeichen
02685  46b9  53 57 41          .byte 'swaP'     $fe $23
02685  46bc  d0
02686  46bd  4f 46 c6          .byte 'ofF'      $fe $24
02687  46c0                _offt  = $24
02688  46c0  46 41 53          .byte 'fasT'     $fe $25
02688  46c3  d4
02689  46c4  53 4c 4f          .byte 'sloW'     $fe $26
02689  46c7  d7
02690  46c8                _hecmt = $26             höchster ESC COMMAND Token
02691  46c8  00                .byte _eom
02692  46c9
02693  46c9
02694  46c9  ===> BASIC ESC FUNCTION Tabelle <===
02695  46c9
02696  46c9  50 4f d4    escfn  .byte 'poT'       $ce $02
02697  46cc                _lefnt = 2             niedrigster ESC FUNCTION Token
```

```
02698  46cc  42 55 4d          .byte 'bumP'      $ce $03
02698  46cf  d0
02699  46d0  50 45 ce          .byte 'peN'       $ce $04
02700  46d3  52 53 50          .byte 'rsppoS'    $ce $05
02700  46d6  50 4f d3
02701  46d9  52 53 50          .byte 'rspritE'   $ce $06
02701  46dc  52 49 54
02701  46df  c5
02702  46e0  52 53 50          .byte 'rspcoloR'  $ce $07
02702  46e3  43 4f 4c
02702  46e6  4f d2
02703  46e8  58 4f d2          .byte 'xoR'       $ce $08
02704  46eb  52 57 49          .byte 'rwindoW'   $ce $09
02704  46ee  4e 44 4f
02704  46f1  d7
02705  46f2  50 4f 49          .byte 'pointeR'   $ce $0a
02705  46f5  4e 54 45
02705  46f8  d2
02706  46f9                _point = $0a
02707  46f9                _hefnt = $0a            höchster ESC FUNCTION Token
02708  46f9
02709  46f9  00                .byte _eom
02710  46fa  00 00             .word 0
02711  46fc
02712  46fc
02713  46fc                    .end
02714  46fc                    .lib basic.dispatch
02715  46fc  put "@basic.dispatch"
```

```
02717  46fc
02718  46fc
02719  46fc  ===> Adreßtabelle der BASIC-Befehle <===
02720  46fc
02721  46fc  cc 4b      stmdsp .word end-1      BASIC-Statement END ($80)
02722  46fe  f8 5d             .word for-1      BASIC-Statement FOR ($81)
02723  4700  f3 57             .word next-1     BASIC-Statement NEXT ($82)
02724  4702  8e 52             .word data-1     BASIC-Statement DATA ($83)
02725  4704  47 56             .word inputn-1   BASIC-Statement INPUT# ($84)
02726  4706  61 56             .word input-1    BASIC-Statement INPUT ($85)
02727  4708  7a 58             .word dim-1      BASIC-Statement DIM ($86)
02728  470a  a8 56             .word read-1     BASIC-Statement READ ($87)
02729  470c  c5 53             .word let-1      BASIC-Statement LET ($88)
02730  470e  da 59             .word goto-1     BASIC-Statement GOTO ($89)
02731  4710  9a 5a             .word run-1      BASIC-Statement RUN ($8a)
02732  4712  c4 52             .word if-1       BASIC-Statement IF ($8b)
02733  4714  c9 5a             .word restor-1   BASIC-Statement RESTORE ($8c)
02734  4716  ce 59             .word gosub-1    BASIC-Statement GOSUB ($8d)
02735  4718  61 52             .word return-1   BASIC-Statement RETURN ($8e)
02736  471a  9c 52             .word rem-1      BASIC-Statement REM ($8f)
02737  471c  ca 4b             .word stop-1     BASIC-Statement STOP ($90)
02738  471e  a2 53             .word ongoto-1   BASIC-Statement ON ($91)
02739  4720  2c 6c             .word wait-1     BASIC-Statement WAIT ($92)
02740  4722  2b 91             .word load-1     BASIC-Statement LOAD ($93)
02741  4724  11 91             .word save-1     BASIC-Statement SAVE ($94)
02742  4726  28 91             .word verify-1   BASIC-Statement VERIFY ($95)
02743  4728  f9 84             .word def-1      BASIC-Statement DEF ($96)
02744  472a  e4 80             .word poke-1     BASIC-Statement POKE ($97)
02745  472c  39 55             .word printn-1   BASIC-Statement PRINT# ($98)
02746  472e  59 55             .word print-1    BASIC-Statement PRINT ($99)
02747  4730  5f 5a             .word cont-1     BASIC-Statement CONT ($9a)
02748  4732  e1 50             .word list-1     BASIC-Statement LIST ($9b)
02749  4734  f7 51             .word clear-1    BASIC-Statement CLR ($9c)
02750  4736  3f 55             .word cmd-1      BASIC-Statement CMD ($9d)
02751  4738  84 58             .word sys-1      BASIC-Statement SYS ($9e)
02752  473a  8c 91             .word open-1     BASIC-Statement OPEN ($9f)
02753  473c  99 91             .word cclos-1    BASIC-Statement CLOSE ($a0)
02754  473e  11 56             .word get-1      BASIC-Statement GET ($a1)
02755  4740  d5 51             .word new-1      BASIC-Statement NEW ($a2)
02756  4742
02757  4742  90 53             .word else-1     BASIC-Statement ELSE ($d5)
02758  4744  61 5f             .word resume-1   BASIC-Statement RESUME ($d6)
02759  4746  4c 5f             .word trap-1     BASIC-Statement TRAP ($d7)
02760  4748  b3 58             .word tron-1     BASIC-Statement TRON ($d8)
02761  474a  b6 58             .word troff-1    BASIC-Statement TROFF ($d9)
02762  474c  eb 71             .word sound-1    BASIC-Statement SOUND ($da)
02763  474e  c4 71             .word volume-1   BASIC-Statement VOL ($db)
02764  4750  74 59             .word auto-1     BASIC-Statement AUTO ($dc)
02765  4752  33 5f             .word puctrl-1   BASIC-Statement PUDEF ($dd)
02766  4754  59 6b             .word grphik-1   BASIC-Statement GRAPHIC ($de)
02767  4756  a7 61             .word paint-1    BASIC-Statement PAINT ($df)
02768  4758  d6 67             .word char-1     BASIC-Statement CHAR ($e0)
02769  475a  b6 62             .word box-1      BASIC-Statement BOX ($e1)
02770  475c  8d 66             .word circle-1   BASIC-Statement CIRCLE ($e2)
02771  475e  8c 65             .word gshape-1   BASIC-Statement GSHAPE ($e3)
02772  4760  2a 64             .word sshape-1   BASIC-Statement SSHAPE ($e4)
02773  4762  96 67             .word draw-1     BASIC-Statement DRAW ($e5)
02774  4764  54 69             .word locate-1   BASIC-Statement LOCATE ($e6)
```

```
02775  4766  e1 69          .word color-1     BASIC-Statement COLOR ($e7)
02776  4768  78 6a          .word scnclr-1    BASIC-Statement SCNCLR ($e8)
02777  476a  5f 69          .word scale-1     BASIC-Statement SCALE ($e9)
02778  476c  85 59          .word help-1      BASIC-Statement HELP ($ea)
02779  476e  df 5f          .word do-1        BASIC-Statement DO ($eb)
02780  4770  89 60          .word loop-1      BASIC-Statement LOOP (ec)
02781  4772  38 60          .word exit-1      BASIC-Statement EXIT ($ed)
02782  4774  7d a0          .word catlog-1    Disk-B.-Statem. DIRECTORY/CATALOG
                                               ($ee/ $fe $0c)
02783  4776  8b a1          .word dsave-1     Disk-B.-Statem. DSAVE ($ef)
02784  4778  a6 a1          .word dload-1     Disk-B.-Statem. DLOAD ($f0)
02785  477a  66 a2          .word header-1    Disk-B.-Statem. HEADER ($f1)
02786  477c  a0 a2          .word scratc-1    Disk-B.-Statem. SCRATCH ($f2)
02787  477e  2e a3          .word colect-1    Disk-B.-Statem. COLLECT ($f3)
02788  4780  45 a3          .word dcopy-1     Disk-B.-Statem. COPY ($f4)
02789  4782  6d a3          .word rename-1    Disk-B.-Statem. RENAME ($f5)
02790  4784  7b a3          .word backup-1    Disk-B.-Statem. BACKUP ($f6)
02791  4786  86 5e          .word delete-1    BASIC-Statement DELETE ($f7)
02792  4788  f7 5a          .word renum-1     BASIC-Statement RENUMBER ($f8)
02793  478a  09 61          .word key-1       BASIC-Statement KEY ($f9)
02794  478c  ff af          .word monit-1     BASIC-Statement MONITOR ($fa)
02795  478e
02796  478e
02797  478e  ===> Adreßtabelle ESC Commands <===
02798  478e
02799  478e  c8 6b    estdsp .word bank-1     DISK-B.-Statement BANK ($fe $02)
02800  4790  45 70          .word filter-1    BASIC-Statement FILTER ($fe $03)
02801  4792  e0 6d          .word play-1      BASIC-Statement PLAY ($fe $04)
02802  4794  d6 6f          .word tempo-1     BASIC-Statement TEMPO ($fe $05)
02803  4796  c5 6c          .word movspr-1    BASIC-Statement MOVSPR ($fe $06)
02804  4798  4e 6c          .word sprite-1    BASIC-Statement SPRITE ($fe $07)
02805  479a  8f 71          .word sprcol-1    BASIC-Statement SPRCOLOR ($fe $08)
02806  479c  bc 58          .word rreg-1      BASIC-Statement RREG ($fe $09)
02807  479e  c0 70          .word envlop-1    BASIC-Statement ENVELOPE ($fe $0a)
02808  47a0  d6 6b          .word sleep-1     BASIC-Statement SLEEP ($fe $0b)
02809  47a2  7d a0          .word catlog-1    Disk-B.-Statem. DIRECTORY/CATALOG
                                               ($ee/ $fe $0c)
02810  47a4  1c a1          .word dopen-1     Disk-B.-Statem. DOPEN ($fe $0d)
02811  47a6  33 a1          .word append-1    Disk-B.-Statem. APPEND ($fe $0e)
02812  47a8  6e a1          .word dclose-1    Disk-B.-Statem. DCLOSE ($fe $0f)
02813  47aa  c7 a1          .word bsave-1     Disk-B.-Statem. BSAVE ($fe $10)
02814  47ac  17 a2          .word bload-1     Disk-B.-Statem. BLOAD ($fe $11)
02815  47ae  d6 a2          .word record-1    Disk-B.-Statem. RECORD ($fe $12)
02816  47b0  61 a3          .word concat-1    Disk-B.-Statem. CONCAT ($fe $13)
02817  47b2  a3 a1          .word dverfy-1    Disk-B.-Statem. DVERIFY ($fe $14)
02818  47b4  21 a3          .word dclear-1    Disk-B.-Statem. DCLEAR ($fe $15)
02819  47b6  eb 76          .word sprsav-1    BASIC-Statement SPRSAV ($fe $16)
02820  47b8  63 71          .word colisn-1    BASIC-Statement COLLISION ($fe $17)
02821  47ba  6b 79          .word snerr-1     BASIC-Statement BEGIN ($fe $18)
02822  47bc  8e 52          .word data-1      BASIC-Statement BEND ($fe $19)
02823  47be  cb 72          .word window-1    BASIC-Statement WINDOW ($fe $1a)
02824  47c0  34 73          .word boot-1      BASIC-Statement BOOT ($fe $1c)
02825  47c2  b5 71          .word setwid-1    BASIC-Statement WIDTH ($fe $1c)
02826  47c4  71 73          .word sprdef-1    BASIC-Statement SPRDEF ($fe $1d)
02827  47c6  45 48          .word badcom-1    BASIC-Statement SYSTEM ($fe $1e)
                                               unimpl.
02828  47c8  1e aa          .word stash-1     BASIC-Statement STASH ($fe $1f)
02829  47ca  00 00          .word 0           $fe $20 = Space
```

```
02830  47cc  23 aa          .word fetch-1    BASIC-Statement FETCH ($fe $21)
02831  47ce  00 00          .word 0          $fe $22 = Anführungszeichen
02832  47d0  28 aa          .word swap-1     BASIC-Statement SWAP ($fe $23)
02833  47d2  45 48          .word badcom-1   BASIC-Statement OFF ($fe $24)
                                              unimpl.
02834  47d4  b2 77          .word fast-1     BASIC-Statement FAST ($fe $25)
02835  47d6  c3 77          .word slow-1     BASIC-Statement SLOW ($fe $26)
02836  47d8
02837  47d8
02838  47d8  ===> Adreßtabelle der BASIC-Funktionen <===
02839  47d8
02840  47d8  65 8c    fundsp .word sgn        BASIC-Funktion SGN ($b4)
02841  47da  fb 8c          .word int        BASIC-Funktion INT ($b5)
02842  47dc  84 8c          .word abs        BASIC-Funktion ABS ($b6)
02843  47de  18 12          .word usrpok     BASIC-Funktion USR ($b7)
02844  47e0  00 80          .word fre        BASIC-Funktion FRE ($b8)
02845  47e2  d0 84          .word pos        BASIC-Funktion POS ($b9)
02846  47e4  b7 8f          .word sqr        BASIC-Funktion SQR ($ba)
02847  47e6  34 84          .word rnd        BASIC-Funktion RND ($bb)
02848  47e8  ca 89          .word log        BASIC-Funktion LOG ($bc)
02849  47ea  33 90          .word exp        BASIC-Funktion EXP ($bd)
02850  47ec  09 94          .word cos        BASIC-Funktion COS ($be)
02851  47ee  10 94          .word sin        BASIC-Funktion SIN ($bf)
02852  47f0  59 94          .word tan        BASIC-Funktion TAN ($c0)
02853  47f2  b3 94          .word atn        BASIC-Funktion ATN ($c1)
02854  47f4  c5 80          .word peek       BASIC-Funktion PEEK ($c2)
02855  47f6  68 86          .word len        BASIC-Funktion LEN ($c3)
02856  47f8  ae 85          .word strd       BASIC-Funktion STR$ ($c4)
02857  47fa  4a 80          .word val        BASIC-Funktion VAL ($c5)
02858  47fc  77 86          .word asc        BASIC-Funktion ASC ($c6)
02859  47fe  bf 85          .word chrd       BASIC-Funktion CHR$ ($c7)
02860  4800  d6 85          .word leftd      BASIC-Funktion LEFT$ ($c8)
02861  4802  0a 86          .word rightd     BASIC-Funktion RIGHT$ ($c9)
02862  4804  1c 86          .word midd       BASIC-Funktion MID$ ($ca)
02863  4806                                  $cb = go
02864  4806  82 81          .word rgr        BASIC-Funktion RGR ($cc)
02865  4808  9b 81          .word rclr       BASIC-Funktion RCLR ($cd)
02866  480a  00 00          .word 0          $ce = ESC FUNCTION Token
02867  480c  03 82          .word joy        BASIC-Funktion JOY ($cf)
02868  480e  0c 9b          .word rdot       BASIC-Funktion RDOT ($d0)
02869  4810  76 80          .word dcml       BASIC-Funktion DEC ($d1)
02870  4812  42 81          .word hexd       BASIC-Funktion HEX$ ($d2)
02871  4814  f6 80          .word errd       BASIC-Funktion ERR$ ($d3)
02872  4816
02873  4816  4d 82          .word pot        BASIC-Funktion POT ($ce $02)
02874  4818  7c 83          .word bump       BASIC-Funktion BUMP ($ce $03)
02875  481a  ae 82          .word pen        BASIC-Funktion PEN ($ce $04)
02876  481c  97 83          .word rsppos     BASIC-Funktion RSPPOS ($ce $05)
02877  481e  1e 83          .word rsprit     BASIC-Funktion RSPRITE ($ce $06)
02878  4820  61 83          .word rspcol     BASIC-Funktion RSPCOLOR ($ce $07)
02879  4822  e1 83          .word xor        BASIC-Funktion XOR ($ce $08)
02880  4824  07 84          .word rwndow     BASIC-Funktion RWINDOW ($ce $09)
02881  4826  fa 82          .word pointr     BASIC-Funktion POINTER ($ce $0a)
02882  4828
02883  4828
```

```
02884  4828  ===> Adresse / Prioritätsflags der math. Routinen <===
02885  4828
02886  4828  79          optab  .byte $79
02887  4829  47 88             .word faddt-1    BASIC-Funktion PLUS ($aa)
02888  482b
02889  482b  79                 .byte $79
02890  482c  30 88             .word fsubt-1    BASIC-Funktion MINUS ($ab)
02891  482e
02892  482e  7b                 .byte $7b
02893  482f  26 8a             .word fmultt-1   BASIC-Funktion MULTIPLIKATION ($ac)
02894  4831
02895  4831  7b                 .byte $7b
02896  4832  4b 8b             .word fdivt-1    BASIC-Funktion DIVISION / ($ad)
02897  4834
02898  4834  7f                 .byte $7f
02899  4835  c0 8f             .word fpwrt-1    BASIC-Funktion POTENZ  ($ae)
02900  4837
02901  4837  50                 .byte $50
02902  4838  88 4c             .word andop-1    BASIC Funktion AND ($af)
02903  483a
02904  483a  46                 .byte $46
02905  483b  85 4c             .word orop-1     BASIC Funktion OR ($b0)
02906  483d
02907  483d  7d                 .byte $7d
02908  483e  f9 8f             .word negop-1    BASIC-Funktion NEG. VORZEICHEN
02909  4840
02910  4840  5a                 .byte $5a
02911  4841  2f 79             .word notop-1    BASIC-Funktion NOT ($a8)
02912  4843
02913  4843  64                 .byte $64
02914  4844  b5 4c             .word dorel-1    BASIC Funktion <=> ($b1-$b3)
02915  4846
02916  4846
02917  4846                     .end
02918  4846                     .lib basic.errlst
02919  4846  put "@basic.errlst"
```

```
02921  4846
02922  4846
02923  4846  ===> Behandlung »?unimplemented command« <===
02924  4846
02925  4846  a2 28      badcom ldx #_erbdc
02926  4848  4c 3c 4d          jmp error        ind. Behandlung Fehlermeldung
02927  484b
02928  484b
02929  484b  ===> Tabelle der Fehlertexte <===
02930  484b
02931  484b             _ertmf = 1
02932  484b  54 4f 4f   errtab .byte 'too many fileS'
02932  484e  20 4d 41
02932  4851  4e 59 20
02932  4854  46 49 4c
02932  4857  45 d3
02933  4859
02934  4859             _erfo  = 2
02935  4859  46 49 4c          .byte 'file opeN'
02935  485c  45 20 4f
02935  485f  50 45 ce
02936  4862
02937  4862             _erfno = 3
02938  4862  46 49 4c          .byte 'file not opeN'
02938  4865  45 20 4e
02938  4868  4f 54 20
02938  486b  4f 50 45
02938  486e  ce
02939  486f
02940  486f             _erfnf = 4
02941  486f  46 49 4c          .byte 'file not founD'
02941  4872  45 20 4e
02941  4875  4f 54 20
02941  4878  46 4f 55
02941  487b  4e c4
02942  487d
02943  487d             _erdnp = 5
02944  487d  44 45 56          .byte 'device not presenT'
02944  4880  49 43 45
02944  4883  20 4e 4f
02944  4886  54 20 50
02944  4889  52 45 53
02944  488c  45 4e d4
02945  488f
02946  488f             _ernif = 6
02947  488f  4e 4f 54          .byte 'not input filE'
02947  4892  20 49 4e
02947  4895  50 55 54
02947  4898  20 46 49
02947  489b  4c c5
02948  489d
02949  489d             _ernof = 7
02950  489d  4e 4f 54          .byte 'not output filE'
02950  48a0  20 4f 55
02950  48a3  54 50 55
02950  48a6  54 20 46
02950  48a9  49 4c c5
02951  48ac
```

```
02952  48ac                 _ermfn = 8
02953  48ac  4d 49 53               .byte 'missing file namE'
02953  48af  53 49 4e
02953  48b2  47 20 46
02953  48b5  49 4c 45
02953  48b8  20 4e 41
02953  48bb  4d c5
02954  48bd
02955  48bd                 _eridn = 9
02956  48bd  49 4c 4c               .byte 'illegal device numbeR'
02956  48c0  45 47 41
02956  48c3  4c 20 44
02956  48c6  45 56 49
02956  48c9  43 45 20
02956  48cc  4e 55 4d
02956  48cf  42 45 d2
02957  48d2
02958  48d2                 _ernwf = 10
02959  48d2  4e 45 58               .byte 'next without foR'
02959  48d5  54 20 57
02959  48d8  49 54 48
02959  48db  4f 55 54
02959  48de  20 46 4f
02959  48e1  d2
02960  48e2
02961  48e2                 _ersn  = 11
02962  48e2  53 59 4e               .byte 'syntaX'
02962  48e5  54 41 d8
02963  48e8
02964  48e8                 _errg  = 12
02965  48e8  52 45 54               .byte 'return without gosuB'
02965  48eb  55 52 4e
02965  48ee  20 57 49
02965  48f1  54 48 4f
02965  48f4  55 54 20
02965  48f7  47 4f 53
02965  48fa  55 c2
02966  48fc
02967  48fc                 _erod  = 13
02968  48fc  4f 55 54               .byte 'out of datA'
02968  48ff  20 4f 46
02968  4902  20 44 41
02968  4905  54 c1
02969  4907
02970  4907                 _erfc  = 14
02971  4907  49 4c 4c               .byte 'illegal quantitY'
02971  490a  45 47 41
02971  490d  4c 20 51
02971  4910  55 41 4e
02971  4913  54 49 54
02971  4916  d9
02972  4917
02973  4917                 _erov  = 15
02974  4917  4f 56 45               .byte 'overfloW'
02974  491a  52 46 4c
02974  491d  4f d7
02975  491f
02976  491f                 _erom  = 16
```

```
02977  491f  4f 55 54          .byte 'out of memorY'
02977  4922  20 4f 46
02977  4925  20 4d 45
02977  4928  4d 4f 52
02977  492b  d9
02978  492c
02979  492c                _erus  = 17
02980  492c  55 4e 44          .byte 'undef''d statemenT'
02980  492f  45 46 27
02980  4932  44 20 53
02980  4935  54 41 54
02980  4938  45 4d 45
02980  493b  4e d4
02981  493d
02982  493d                _erbs  = 18
02983  493d  42 41 44          .byte 'bad subscripT'
02983  4940  20 53 55
02983  4943  42 53 43
02983  4946  52 49 50
02983  4949  d4
02984  494a
02985  494a                _erdd  = 19
02986  494a  52 45 44          .byte 'redim''d arraY'
02986  494d  49 4d 27
02986  4950  44 20 41
02986  4953  52 52 41
02986  4956  d9
02987  4957
02988  4957                _erdv0 = 20
02989  4957  44 49 56          .byte 'division by zerO'
02989  495a  49 53 49
02989  495d  4f 4e 20
02989  4960  42 59 20
02989  4963  5a 45 52
02989  4966  cf
02990  4967
02991  4967                _erid  = 21
02992  4967  49 4c 4c          .byte 'illegal direcT'
02992  496a  45 47 41
02992  496d  4c 20 44
02992  4970  49 52 45
02992  4973  43 d4
02993  4975
02994  4975                _ertm  = 22
02995  4975  54 59 50          .byte 'type mismatcH'
02995  4978  45 20 4d
02995  497b  49 53 4d
02995  497e  41 54 43
02995  4981  c8
02996  4982
02997  4982                _erls  = 23
02998  4982  53 54 52          .byte 'string too lonG'
02998  4985  49 4e 47
02998  4988  20 54 4f
02998  498b  4f 20 4c
02998  498e  4f 4e c7
02999  4991
03000  4991                _erbd  = 24
```

```
03001  4991  46 49 4c          .byte 'file datA'
03001  4994  45 20 44
03001  4997  41 54 c1
03002  499a
03003  499a                _erst  = 25
03004  499a  46 4f 52          .byte 'formula too compleX'
03004  499d  4d 55 4c
03004  49a0  41 20 54
03004  49a3  4f 4f 20
03004  49a6  43 4f 4d
03004  49a9  50 4c 45
03004  49ac  d8
03005  49ad
03006  49ad                _ercn  = 26
03007  49ad  43 41 4e          .byte 'can''t continuE'
03007  49b0  27 54 20
03007  49b3  43 4f 4e
03007  49b6  54 49 4e
03007  49b9  55 c5
03008  49bb
03009  49bb                _eruf  = 27
03010  49bb  55 4e 44          .byte 'undef''d functioN'
03010  49be  45 46 27
03010  49c1  44 20 46
03010  49c4  55 4e 43
03010  49c7  54 49 4f
03010  49ca  ce
03011  49cb
03012  49cb                _ervfy = 28
03013  49cb  56 45 52          .byte 'verifY'
03013  49ce  49 46 d9
03014  49d1
03015  49d1                _erlod = 29
03016  49d1  4c 4f 41          .byte 'loaD'
03016  49d4  c4
03017  49d5
03018  49d5                _erbrk = 30
03019  49d5  42 52 45          .byte 'break', _eom, $a0 _eom := ohne '_eror'
03019  49d8  41 4b 00
03019  49db  a0
03020  49dc
03021  49dc                _ercr  = 31
03022  49dc  43 41 4e          .byte 'can''t resumE'
03022  49df  27 54 20
03022  49e2  52 45 53
03022  49e5  55 4d c5
03023  49e8
03024  49e8                _erlnf = 32
03025  49e8  4c 4f 4f          .byte 'loop not founD'
03025  49eb  50 20 4e
03025  49ee  4f 54 20
03025  49f1  46 4f 55
03025  49f4  4e c4
03026  49f6
03027  49f6                _erlwd = 33
03028  49f6  4c 4f 4f          .byte 'loop without dO'
03028  49f9  50 20 57
03028  49fc  49 54 48
```

```
03028  49ff  4f 55 54
03028  4a02  20 44 cf
03029  4a05
03030  4a05              _eroid = 34
03031  4a05  44 49 52           .byte 'direct mode onlY'
03031  4a08  45 43 54
03031  4a0b  20 4d 4f
03031  4a0e  44 45 20
03031  4a11  4f 4e 4c
03031  4a14  d9
03032  4a15
03033  4a15              _erng  = 35
03034  4a15  4e 4f 20           .byte 'no graphics areA'
03034  4a18  47 52 41
03034  4a1b  50 48 49
03034  4a1e  43 53 20
03034  4a21  41 52 45
03034  4a24  c1
03035  4a25
03036  4a25              _erbdk = 36
03037  4a25  42 41 44           .byte 'bad disK'
03037  4a28  20 44 49
03037  4a2b  53 cb
03038  4a2d
03039  4a2d              _ernob = 37
03040  4a2d  42 45 4e           .byte 'bend not founD'
03040  4a30  44 20 4e
03040  4a33  4f 54 20
03040  4a36  46 4f 55
03040  4a39  4e c4
03041  4a3b
03042  4a3b              _erntl = 38
03043  4a3b  4c 49 4e           .byte 'line number too largE'
03043  4a3e  45 20 4e
03043  4a41  55 4d 42
03043  4a44  45 52 20
03043  4a47  54 4f 4f
03043  4a4a  20 4c 41
03043  4a4d  52 47 c5
03044  4a50
03045  4a50              _erref = 39
03046  4a50  55 4e 52           .byte 'unresolved referencE'
03046  4a53  45 53 4f
03046  4a56  4c 56 45
03046  4a59  44 20 52
03046  4a5c  45 46 45
03046  4a5f  52 45 4e
03046  4a62  43 c5
03047  4a64
03048  4a64              _erbdc = 40
03049  4a64  55 4e 49           .byte 'unimplemented commanD'
03049  4a67  4d 50 4c
03049  4a6a  45 4d 45
03049  4a6d  4e 54 45
03049  4a70  44 20 43
03049  4a73  4f 4d 4d
03049  4a76  41 4e c4
03050  4a79
```

```
03051  4a79                _erlst = 41
03052  4a79  46 49 4c       .byte 'file reaD'
03052  4a7c  45 20 52
03052  4a7f  45 41 c4
03053  4a82
03054  4a82
03055  4a82  ===> index2 auf Fehlertext (Nummer in xr) einstellen <===
03056  4a82
03057  4a82  aa          erstup tax              Zähler Fehlertexte
03058  4a83  a0 00              ldy #0
03059  4a85
03060  4a85  a9 4b              lda #<errtab
03061  4a87  85 26              sta index2       ADDRL Zeiger in Fehlertexte
03062  4a89
03063  4a89  a9 48              lda #>errtab
03064  4a8b  85 27              sta index2+1     ADDRH Zeiger in Fehlertexte
03065  4a8d
03066  4a8d
03067  4a8d  ===> Fehlertexte zählen <===
03068  4a8d
03069  4a8d  ca          erst10 dex              Zähler Fehlertexte
03070  4a8e  30 0e              bmi erst40       index2 auf Fehlertext eingestellt
03071  4a90
03072  4a90
03073  4a90  ===> Ende Fehlertext suchen <===
03074  4a90
03075  4a90  b1 26      erst20 lda (index2),y   Zeichen aus Fehlertext
03076  4a92  48               pha
03077  4a93
03078  4a93  e6 26              inc index2       Zeiger auf
03079  4a95  d0 02              bne erst30       nächstes Zeichen
03080  4a97  e6 27              inc index2+1     einstellen
03081  4a99
03082  4a99  68          erst30 pla              Bit7 testen:
03083  4a9a  10 f4              bpl erst20       Ende Fehlertext suchen
03084  4a9c  30 ef              bmi erst10       Fehlertexte zählen
03085  4a9e
03086  4a9e
03087  4a9e  ===> index2 auf Fehlertext eingestellt <===
03088  4a9e
03089  4a9e  60          erst40 rts
03090  4a9f
03091  4a9f
03092  4a9f                     .end
03093  4a9f                     .lib basic.exec
03094  4a9f  put "@basic.exec"
```

```
03096  4a9f
03097  4a9f
03098  4a9f  ===> Test BASIC IRQ, Token auswerten <===
03099  4a9f
03100  4a9f  6c 08 03   xeqcm  jmp (igone)      ind. Token auswerten ($4aa2)
03101  4aa2
03102  4aa2
03103  4aa2  ===> Test BASIC IRQ, Token auswerten <===
03104  4aa2
03105  4aa2  24 7f       ngone  bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
03106  4aa4  10 4a              bpl gone30
03107  4aa6
03108  4aa6  ad 7f 12           lda intval       Flag für Bearbeitung COLLISION
03109  4aa9
03110  4aa9   7.......  1 = COLLISION wird bearbeitet
03111  4aa9
03112  4aa9  30 45              bmi gone30
03113  4aab
03114  4aab  ===> Sprite-Kollisionen bearbeiten <===
03115  4aab
03116  4aab  a2 02              ldx #2           Zähler für Typen COLLISION
03117  4aad  bd 76 12   gone10  lda tripfg,x     Flag für Bearbeitung Typ
03118  4ab0  f0 3b              beq gone20       nicht gesetzt
03119  4ab2
03120  4ab2  a9 00              lda #0
03121  4ab4  9d 76 12           sta tripfg,x     Flag zurücksetzen
03122  4ab7
03123  4ab7  bd 79 12           lda intadl,x     Tab. Low-Byte Zeilennr. COLISN
03124  4aba  85 16              sta linnum
03125  4abc  bd 7c 12           lda intadh,x     Tab. High-Byte Zeilennr. COLISN
03126  4abf  85 17              sta linnum+1
03127  4ac1
03128  4ac1  8a                 txa
03129  4ac2  48                 pha              Typ COLLISION
03130  4ac3
03131  4ac3  a5 3d              lda txtptr       ADDRL chrget-Zeiger
03132  4ac5  48                 pha
03133  4ac6
03134  4ac6  a5 3e              lda txtptr+1     ADDRH chrget-Zeiger
03135  4ac8  48                 pha
03136  4ac9
03137  4ac9  ad 7f 12           lda intval       Flag für Bearbeitung COLLISION
03138  4acc  09 80              ora #%10000000
03139  4ace
03140  4ace   7.......  1 = COLLISION wird bearbeitet
03141  4ace
03142  4ace  8d 7f 12           sta intval       Flag für Bearbeitung COLLISION
03143  4ad1
03144  4ad1  20 80 03           jsr chrget       nächstes Zeichen nach ac
03145  4ad4  20 1d 5a           jsr gosusb       Textzeiger, Zeilennummer und
                                                 gosub-Token auf Stapel
03146  4ad7  20 e2 59           jsr goto_1       GOSUB COLLISION
03147  4ada  20 f6 4a           jsr newstt       Stopptaste abfragen, nächst.
                                                 BASIC-Befehl/Zeile
03148  4add
03149  4add  ad 7f 12           lda intval       Flag für Bearbeitung COLLISION
03150  4ae0  29 7f              and #%01111111
03151  4ae2
```

```
03152  4ae2   7.......  0 = COLLISION nicht in Arbeit
03153  4ae2
03154  4ae2  8d 7f 12          sta intval       Flag für Bearbeitung COLLISION
03155  4ae5
03156  4ae5  68                pla
03157  4ae6  85 3e             sta txtptr+1     ADDRH chrget-Zeiger
03158  4ae8
03159  4ae8  68                pla
03160  4ae9  85 3d             sta txtptr       ADDRL chrget-Zeiger
03161  4aeb
03162  4aeb  68                pla              Typ COLLISION
03163  4aec  aa                tax
03164  4aed
03165  4aed  ca          gone20 dex
03166  4aee  10 bd             bpl gone10
03167  4af0
03168  4af0  20 80 03    gone30 jsr chrget       nächstes Zeichen nach ac
03169  4af3
03170  4af3
03171  4af3  ===> Token auswerten u. BASIC-Routine aufrufen <===
03172  4af3
03173  4af3  20 3f 4b    xeqdir jsr xeqcm1       BASIC-Befehl (Token in ac) ausführen
03174  4af6
03175  4af6
03176  4af6  ===> Stopptaste abfragen, nächst. BASIC-Befehl/Zeile <===
03177  4af6
03178  4af6  20 b5 4b    newstt jsr istpky       Stopp-Taste abfragen
03179  4af9
03180  4af9  24 7f             bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
03181  4afb  10 06             bpl nstt10
03182  4afd
03183  4afd  ===> chrget- und Stapelzeiger retten <===
03184  4afd
03185  4afd  20 34 4b          jsr tto          txtptr nach oldtxt
03186  4b00  ba                tsx
03187  4b01  86 82             stx oldstk       Stapelzeiger für trap
03188  4b03
03189  4b03  a0 00       nstt10 ldy #0
03190  4b05  20 c9 03          jsr indtxt       LDA (txtptr),y aus PGM-RAM
03191  4b08  f0 03             beq nstt20       Ende Programmzeile gefunden ($00)
03192  4b0a
03193  4b0a  4c ae 4b          jmp morsts       Test auf weiteren BASIC-Befehl in
                                                 dieser Zeile
03194  4b0d
03195  4b0d
03196  4b0d  ===> Test auf PGM-Ende, Parameter nächste Zeile <===
03197  4b0d
03198  4b0d  24 7f       nstt20 bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
03199  4b0f  10 20             bpl nstt40       Ende Befehlsfolge im Direktmodus,
                                                 ready
03200  4b11
03201  4b11  a0 02             ldy #2           Zeiger auf ADDRH Link
03202  4b13  20 c9 03          jsr indtxt       LDA (txtptr),y aus PGM-RAM
03203  4b16  f0 19             beq nstt40       Ende Programm gefunden (ADDRH=0)
03204  4b18
03205  4b18  c8                iny              Zeiger auf Low-Byte Zeilennummer
03206  4b19  20 c9 03          jsr indtxt       LDA (txtptr),y aus PGM-RAM
03207  4b1c  85 3b             sta curlin       Low-Byte akt. Zeilennummer
```

```
03208  4b1e
03209  4b1e  c8              iny             Zeiger auf High-Byte Zeilennummer
03210  4b1f  20 c9 03        jsr indtxt      LDA (txtptr),y aus PGM-RAM
03211  4b22  85 3c           sta curlin+1    High-Byte akt. Zeilennummer
03212  4b24
03213  4b24  ===> chrget-Zeiger hinter Zeilennummer <===
03214  4b24
03215  4b24  98              tya
03216  4b25  18              clc
03217  4b26  65 3d           adc txtptr      ADDRL chrget-Zeiger
03218  4b28  85 3d           sta txtptr      ADDRL chrget-Zeiger
03219  4b2a
03220  4b2a  90 02           bcc nstt30      nächste Programmzeile abarbeiten
03221  4b2c  e6 3e           inc txtptr+1    ADDRH chrget-Zeiger
03222  4b2e
03223  4b2e  4c 9f 4a  nstt30 jmp xeqcm      nächste Programmzeile abarbeiten
03224  4b31
03225  4b31
03226  4b31  4c 37 4d  nstt40 jmp ready      Ende Befehlsfolge im Direktmodus,
                                             ready
03227  4b34
03228  4b34
03229  4b34  ===> txtptr nach oldtxt <===
03230  4b34
03231  4b34  a5 3d      tto    lda txtptr     ADDRL chrget-Zeiger
03232  4b36  a4 3e             ldy txtptr+1   ADDRH chrget-Zeiger
03233  4b38
03234  4b38  8d 02 12          sta oldtxt     ADDRL Zeiger letzter BASIC-Befehl
03235  4b3b  8c 03 12          sty oldtxt+1   ADDRH Zeiger letzter BASIC-Befehl
03236  4b3e
03237  4b3e  60        xeqrts rts
03238  4b3f
03239  4b3f
03240  4b3f  ===> BASIC-Befehl (Token in ac) ausführen <===
03241  4b3f
03242  4b3f  f0 fd      xeqcm1 beq xeqrts      Zeilenende gefunden
03243  4b41
03244  4b41  2c 6f 11          bit trcflg     TRACE-Flag (0=aus, ff=ein)
03245  4b44  10 13             bpl xeqcm2     TRACE aus
03246  4b46
03247  4b46  24 7f             bit runmod     Modus: $80=PGM, $40=OVL, $00=DIR
03248  4b48  10 0f             bpl xeqcm2     nicht Programm-Modus
03249  4b4a
03250  4b4a  ===> Ausgabe <Zeilennummer> für TRACE <===
03251  4b4a
03252  4b4a  48                pha            akt. BASIC-Token
03253  4b4b
03254  4b4b  a9 5b             lda #$5b       ASCII für »eckige Klammer auf«
03255  4b4d  20 0c 56          jsr outdo      Ausgabe ac auf akt. Kanal
03256  4b50
03257  4b50  20 2e 8e          jsr curprt     Ausgabe akt. Zeilennummer
03258  4b53
03259  4b53  a9 5d             lda #$5d       ASCII für »eckige Klammer zu«
03260  4b55  20 0c 56          jsr outdo      Ausgabe ac auf akt. Kanal
03261  4b58
03262  4b58  68                pla            akt. BASIC-Token
03263  4b59
03264  4b59
```

```
03265  4b59  ===> BASIC-Token prüfen <===
03266  4b59
03267  4b59  c9 fe      xeqcm2 cmp #_ecmdt      ESC COMMAND Token
03268  4b5b  f0 37             beq xeqesc      ESC COMMAND Token bearbeiten
03269  4b5d
03270  4b5d  c9 cb             cmp #_got       GO Token
03271  4b5f  d0 03             bne xeqcm3      Zeiger in Adreßtabelle berechnen
03272  4b61
03273  4b61  4c 3d 5a          jmp gowoto      BASIC-Statement GO ($cb)
03274  4b64
03275  4b64
03276  4b64  ===> Zeiger in Adreßtabelle berechnen <===
03277  4b64
03278  4b64  c9 ca      xeqcm3 cmp #_middt     MID$ Token
03279  4b66  f0 23             beq xeqmid      Adresse von mid$ auf Stapel,
                                               ausführen
03280  4b68
03281  4b68  c9 fb             cmp #_usint     USING Token
03282  4b6a  b0 3f             bcs snerr1      Behandlung »?syntax error«
03283  4b6c
03284  4b6c  ===> zwischen Token $a4 u. $d5 Funktionen <===
03285  4b6c
03286  4b6c  c9 a3             cmp #_tabt      TAB( Token
03287  4b6e  90 06             bcc xeqcm4      Test auf BASIC-Statement LET
03288  4b70
03289  4b70  c9 d5             cmp #_elset     ELSE Token
03290  4b72  90 37             bcc snerr1      Behandlung »?syntax error«
03291  4b74
03292  4b74  e9 32             sbc #_elset-_tabt
03293  4b76
03294  4b76
03295  4b76  ===> Test auf BASIC-Statement LET <===
03296  4b76
03297  4b76  38        xeqcm4 sec
03298  4b77  e9 80             sbc #_tkofs
03299  4b79  b0 03             bcs xeqcm5      Adresse BASIC-Routine auf Stapel,
                                               ausführen
03300  4b7b
03301  4b7b  4c c6 53          jmp let         BASIC-Statement LET ($88)
03302  4b7e
03303  4b7e
03304  4b7e  ===> Adresse BASIC-Routine auf Stapel, ausführen <===
03305  4b7e
03306  4b7e  0a        xeqcm5 asl a
03307  4b7f  a8               tay
03308  4b80  b9 fd 46         lda stmdsp+1,y  Adreßtabelle der BASIC-Befehle
03309  4b83  48               pha
03310  4b84
03311  4b84  b9 fc 46         lda stmdsp,y    Adreßtabelle der BASIC-Befehle
03312  4b87  48               pha
03313  4b88
03314  4b88  4c 80 03         jmp chrget      nächstes Zeichen nach ac
03315  4b8b
03316  4b8b
03317  4b8b  ===> Adresse von mid$ auf Stapel, ausführen <===
03318  4b8b
03319  4b8b  a9 59      xeqmid lda #>replc-1   Zuweisung MID$
03320  4b8d  48               pha
```

```
03321  4b8e
03322  4b8e  a9 00           lda #<replc-1    Zuweisung MID$
03323  4b90  48              pha
03324  4b91
03325  4b91  4c 80 03  xeqchr jmp chrget      nächstes Zeichen nach ac
03326  4b94
03327  4b94
03328  4b94
03329  4b94  ===> ESC COMMAND Token bearbeiten <===
03330  4b94
03331  4b94  20 80 03  xeqesc jsr chrget      nächstes Zeichen nach ac
03332  4b97  f0 12           beq snerr1       Behandlung »?syntax error«
03333  4b99
03334  4b99  c9 02           cmp #_lecmt      niedrigster ESC COMMAND Token
03335  4b9b  90 08           bcc xeqes1       ind. Sprung zu user-def.
                                              BASIC-Erweiterung
03336  4b9d
03337  4b9d  c9 27           cmp #_hecmt+1    höchster ESC COMMAND Token
03338  4b9f  b0 04           bcs xeqes1       ind. Sprung zu user-def.
                                              BASIC-Erweiterung
03339  4ba1
03340  4ba1  69 47           adc #estdsp-stmdsp/2-_lecmt
03341  4ba3  d0 d9           bne xeqcm5       immer
03342  4ba5
03343  4ba5
03344  4ba5  ===> ind. Sprung zu user-def. BASIC-Erweiterung <===
03345  4ba5
03346  4ba5  38        xeqes1 sec              Fehler Flag
03347  4ba6  6c 10 03         jmp (iescex)     ind. Routine zu user-def. Token
                                              aufr. ($4ba9)
03348  4ba9
03349  4ba9
03350  4ba9  ===> Routine zu user-def. Token aufrufen <===
03351  4ba9
03352  4ba9  90 e6     nescex bcc xeqchr
03353  4bab
03354  4bab  ===> Behandlung »?syntax error« <===
03355  4bab
03356  4bab  4c 6c 79  snerr1 jmp snerr        Behandlung »?syntax error«
03357  4bae
03358  4bae
03359  4bae  ===> Test auf weiteren BASIC-Befehl in dieser Zeile <===
03360  4bae
03361  4bae  c9 3a     morsts cmp #':'
03362  4bb0  d0 f9           bne snerr1       Behandlung »?syntax error«
03363  4bb2
03364  4bb2  4c 9f 4a        jmp xeqcm        ind. Test BASIC-IRQ, Token
                                              Auswertung
03365  4bb5
03366  4bb5
03367  4bb5  ===> Stopp-Taste abfragen <===
03368  4bb5
03369  4bb5  20 93 92  istpky jsr cstop       Abfrage Stopp-Taste, dann Ersatz-I/O
03370  4bb8  f0 01           beq stopc1       Stopp-Taste gedrückt
03371  4bba
03372  4bba  60              rts
03373  4bbb
03374  4bbb
```

```
03375  4bbb  ===> Stopp-Taste gedrückt <===
03376  4bbb
03377  4bbb  ac 0c 12   stopc1 ldy trapno+1     high Zeilennr. TRAP Routine
03378  4bbe  c8               iny
03379  4bbf  f0 0f            beq stop1        im PGM-Modus akt. Zeilennummer nach
                                               oldlin
03380  4bc1
03381  4bc1
03382  4bc1  ===> Abbruch TRAP-Modus aus <===
03383  4bc1
03384  4bc1  20 93 92   stopc0 jsr cstop       Abfrage Stopp-Taste, dann Ersatz-I/O
03385  4bc4  f0 fb            beq stopc0       Abbruch TRAP-Modus aus
03386  4bc6
03387  4bc6  ===> Behandlung »?break« <===
03388  4bc6
03389  4bc6  a2 1e            ldx #_erbrk
03390  4bc8  4c 3c 4d         jmp error        ind. Behandlung Fehlermeldung
03391  4bcb
03392  4bcb
03393  4bcb  ===> BASIC-Statement STOP ($90) <===
03394  4bcb
03395  4bcb  b0 01      stop   bcs stopc
03396  4bcd
03397  4bcd
03398  4bcd  ===> BASIC-Statement END ($80) <===
03399  4bcd
03400  4bcd  18         end    clc
03401  4bce  d0 26      stopc  bne dorts
03402  4bd0
03403  4bd0
03404  4bd0  ===> im PGM-Modus akt. Zeilennummer nach oldlin <===
03405  4bd0
03406  4bd0  24 7f      stop1  bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
03407  4bd2  10 0d            bpl diris        Stapel korrigieren, Ausgabe ready
                                               bzw. break
03408  4bd4
03409  4bd4  20 34 4b         jsr tto          txtptr nach oldtxt
03410  4bd7
03411  4bd7  a5 3b            lda curlin       Low-Byte akt. Zeilennummer
03412  4bd9  a4 3c            ldy curlin+1     High-Byte akt. Zeilennummer
03413  4bdb
03414  4bdb  8d 00 12         sta oldlin       Low-Byte vorige Zeilennummer (stop)
03415  4bde  8c 01 12         sty oldlin+1     High-Byte vorige Zeilennummer (stop)
03416  4be1
03417  4be1
03418  4be1  ===> Stapel korrigieren, Ausgabe ready bzw. break <===
03419  4be1
03420  4be1  68         diris  pla
03421  4be2  68               pla
03422  4be3  90 0e            bcc nsxx6        Sprung nach »ready«
03423  4be5  20 81 92         jsr cprimm       folg. Text ausg./überspringen
03424  4be8
03425  4be8  0d 0a 42         .byte _cr, _lf, 'break', _eom
03425  4beb  52 45 41
03425  4bee  4b 00
03426  4bf0
03427  4bf0  4c af 4d         jmp errfin       Test PGM-Modus, Ausg. Zeilennr.
03428  4bf3
```

```
03429  4bf3
03430  4bf3  4c 37 4d   nsxx6  jmp ready        Sprung nach »ready«
03431  4bf6
03432  4bf6
03433  4bf6  60         dorts  rts
03434  4bf7
03435  4bf7
03436  4bf7  ===> BASIC-Funktionen bearbeiten <===
03437  4bf7
03438  4bf7  c9 ce      isfun  cmp #_efunt       ESC FUNCTION Token
03439  4bf9  f0 59             beq doescf       Aufruf erweit. BASIC-Funktion
03440  4bfb
03441  4bfb  c9 d5             cmp #_elset      ELSE Token
03442  4bfd  b0 ac             bcs snerr1       Behandlung »?syntax error«
03443  4bff
03444  4bff  c9 cb             cmp #_got        GO Token
03445  4c01  90 02      isfn10 bcc isfn20
03446  4c03  e9 01             sbc #1           GO Token nicht in Adreßverteiler
03447  4c05
03448  4c05  48         isfn20 pha              Funktions Token
03449  4c06  aa                tax              Funktions Token
03450  4c07  20 80 03          jsr chrget       nächstes Zeichen nach ac
03451  4c0a
03452  4c0a  e0 d3             cpx #_instt-1    INSTR Token
03453  4c0c  f0 08             beq isfn30       Bearbeitung LEFT$, MID$, RIGHT$,
                                                 INSTR
03454  4c0e
03455  4c0e  e0 cb             cpx #_middt+1    MID$ Token
03456  4c10  b0 29             bcs oknorm
03457  4c12
03458  4c12  e0 c8             cpx #_leftt      LEFT$ Token
03459  4c14  90 25             bcc oknorm
03460  4c16
03461  4c16
03462  4c16  ===> Bearbeitung LEFT$, MID$, RIGHT$, INSTR <===
03463  4c16
03464  4c16  20 59 79   isfn30 jsr chkopn       Test auf folg. '(', sonst Fehler
03465  4c19  20 ef 77          jsr frmevl       folgenden Ausdruck auswerten
03466  4c1c  20 5c 79          jsr chkcom       Test auf folg. »Komma«, sonst Fehler
03467  4c1f  20 dd 77          jsr chkstr       Test String VAR, sonst type mism.
03468  4c22
03469  4c22  68                pla              Funktions Token
03470  4c23
03471  4c23  c9 d3             cmp #_instt-1    INSTR Token
03472  4c25  f0 59             beq instgo       Aufruf Funktion instr
03473  4c27
03474  4c27  ===> Behandlung LEFT$, MID$, RIGHT$ <===
03475  4c27
03476  4c27  aa                tax              Funktions Token
03477  4c28
03478  4c28  a5 67             lda indice+1     ADDRH Zeiger auf Stringdescriptor
03479  4c2a  48                pha
03480  4c2b
03481  4c2b  a5 66             lda indice       ADDRL Zeiger auf Stringdescriptor
03482  4c2d  48                pha
03483  4c2e
03484  4c2e  8a                txa              Funktions Token
03485  4c2f  48                pha
```

```
03486  4c30
03487  4c30  20 f4 87          jsr getbyt        8 Bit INT üb. frmnum nach xr
03488  4c33
03489  4c33  68               pla               Funktions Token
03490  4c34  a8               tay
03491  4c35
03492  4c35  8a               txa               Parameter getbyt
03493  4c36  48               pha
03494  4c37
03495  4c37  98               tya               Funktions Token
03496  4c38  4c 3f 4c         jmp fingo         Adresse Funktionsroutine eintragen,
                                                 Aufruf
03497  4c3b
03498  4c3b
03499  4c3b  20 50 79   oknorm jsr parchk       Test auf folg.'(' ')', sonst Fehler
03500  4c3e  68               pla               Funktions Token
03501  4c3f
03502  4c3f
03503  4c3f  ===> Adresse Funktionsroutine eintragen, Aufruf <===
03504  4c3f
03505  4c3f  38         fingo  sec
03506  4c40  e9 b4             sbc #_onefn       kleinster Funktions Token
03507  4c42  0a               asl a             mal zwei
03508  4c43  a8               tay               Zeiger in Adreßtabelle
03509  4c44
03510  4c44  b9 d9 47          lda fundsp+1,y    Adreßtabelle der BASIC-Funktionen
03511  4c47  85 58             sta jmper+2       ADDRH Funktions Routine
03512  4c49
03513  4c49  b9 d8 47          lda fundsp,y      Adreßtabelle der BASIC-Funktionen
03514  4c4c  85 57             sta jmper+1       ADDRL Funktions Routine
03515  4c4e
03516  4c4e  20 56 00          jsr jmper         Funktions Routine aufrufen
03517  4c51  4c da 77          jmp chknum        Test num. VAR, sonst type mism.
03518  4c54
03519  4c54
03520  4c54  ===> Aufruf erweit. BASIC-Funktion <===
03521  4c54
03522  4c54  20 80 03   doescf jsr chrget        Funktions Token nach ac
03523  4c57  f0 2a             beq snerr6        Trennzeichen
03524  4c59
03525  4c59  c9 0a             cmp #_point       POINTER Token
03526  4c5b  f0 0b             beq doesc1
03527  4c5d
03528  4c5d  48               pha               Funktions Token
03529  4c5e
03530  4c5e  20 80 03          jsr chrget        nächstes Zeichen nach ac
03531  4c61  20 59 79          jsr chkopn        Test auf folg. '(', sonst Fehler
03532  4c64  20 ef 77          jsr frmevl        folgenden Ausdruck auswerten
03533  4c67
03534  4c67  68               pla               Funktions Token
03535  4c68
03536  4c68  c9 02      doesc1 cmp #_lefnt       kleinster ESC FUNCTION Token
03537  4c6a  90 08             bcc foregn        Test auf Aufruf user-def. Funktion
03538  4c6c
03539  4c6c  c9 0b             cmp #_hefnt+1     höchster ESC FUNCTION Token
03540  4c6e  b0 04             bcs foregn        Test auf Aufruf user-def. Funktion
03541  4c70
03542  4c70  69 d1             adc #_hofnt-3     höchster alter Funktions Token
```

```
03543  4c72  d0 cb            bne fingo        immer
03544  4c74
03545  4c74
03546  4c74  ===> Test auf Aufruf user-def. Funktion <===
03547  4c74
03548  4c74  38         foregn sec
03549  4c75  20 7d 4c          jsr goforn       Aufruf user-def. Funktion
03550  4c78
03551  4c78
03552  4c78  ===> »?syntax error« user-def. Funktion <===
03553  4c78
03554  4c78  b0 09      nescfn bcs snerr6       Behandlung »?syntax error«
03555  4c7a  4c da 77          jmp chknum       Test num. VAR, sonst type mism.
03556  4c7d
03557  4c7d
03558  4c7d  ===> ind. Aufruf user-def. Funktion <===
03559  4c7d
03560  4c7d  6c fc 02   goforn jmp (efnvec)     ind. Aufruf user-def. Funktion
                                                ($4c78)
03561  4c80
03562  4c80
03563  4c80  ===> Aufruf Funktion INSTR <===
03564  4c80
03565  4c80  4c c1 99   instgo jmp instr        Aufruf Funktion instr
03566  4c83
03567  4c83
03568  4c83  ===> Behandlung »?syntax error« <===
03569  4c83
03570  4c83  4c 6c 79   snerr6 jmp snerr        Behandlung »?syntax error«
03571  4c86
03572  4c86
03573  4c86  ===> BASIC Funktion OR ($b0) <===
03574  4c86
03575  4c86  a0 ff      orop   ldy #%11111111
03576  4c88  2c                .byte _skip2
03577  4c89
03578  4c89
03579  4c89  ===> BASIC Funktion AND ($af) <===
03580  4c89
03581  4c89  a0 00      andop  ldy #%00000000
03582  4c8b  84 0d             sty fconc
03583  4c8d
03584  4c8d  20 b4 84          jsr ayint        Wandlung FAC nach sgnd. INT
03585  4c90
03586  4c90  a5 66             lda facmo        FAC: M2
03587  4c92  45 0d             eor fconc
03588  4c94  85 09             sta charac       XOR: M2
03589  4c96
03590  4c96  a5 67             lda faclo        FAC: M1
03591  4c98  45 0d             eor fconc
03592  4c9a  85 0a             sta endchr       XOR: M1
03593  4c9c
03594  4c9c  20 28 8c          jsr movfa        ARG nach FAC
03595  4c9f  20 b4 84          jsr ayint        Wandlung FAC nach sgnd. INT
03596  4ca2
03597  4ca2  a5 67             lda faclo        FAC: M1
03598  4ca4  45 0d             eor fconc
03599  4ca6  25 0a             and endchr       XOR: M1
```

```
03600  4ca8  45 0d           eor fconc
03601  4caa  a8              tay
03602  4cab
03603  4cab  a5 66           lda facmo       FAC: M2
03604  4cad  45 0d           eor fconc
03605  4caf  25 09           and charac      XOR: M2
03606  4cb1  45 0d           eor fconc
03607  4cb3  4c 3c 79        jmp givayv      16 Bit INT yr/ac nach GK in FAC
03608  4cb6
03609  4cb6
03610  4cb6  ===> BASIC Funktion <=> ($b1-$b3) <===
03611  4cb6
03612  4cb6  20 de 77  dorel  jsr chkval      Test VAR Typ (CARRY=1:String,
                                              =0:num)
03613  4cb9  b0 13           bcs strcmp      Funktion <=> für Alphawerte
03614  4cbb
03615  4cbb  a5 6f           lda argsgn      ARG: SG
03616  4cbd  09 7f           ora #%01111111
03617  4cbf  25 6b           and argho       ARG: M4
03618  4cc1  85 6b           sta argho       ARG: M4
03619  4cc3
03620  4cc3  a9 6a           lda #<argexp    ARG: EX
03621  4cc5  a0 00           ldy #>argexp    ARG: EX
03622  4cc7  20 87 8c        jsr fcomp       FAC mit Var. ac/yr vergleichen
03623  4cca  aa              tax
03624  4ccb  4c 01 4d        jmp qcomp       Wahrheitswert nach fac
03625  4cce
03626  4cce
03627  4cce  ===> Funktion <=> für Alphawerte <===
03628  4cce
03629  4cce  a9 00     strcmp lda #0
03630  4cd0  85 0f           sta valtyp      Flag Variablentyp
03631  4cd2
03632  4cd2  c6 4f           dec opmask      vom akt. Operator erzeugte Bit-Maske
03633  4cd4
03634  4cd4  20 81 87        jsr frefac      Daten Arbeitsstring aus Ablage,
                                              Adresse index1, Länge ac
03635  4cd7  85 63           sta dsctmp
03636  4cd9  86 64           stx dsctmp+1
03637  4cdb  84 65           sty dsctmp+2
03638  4cdd
03639  4cdd  a5 6d           lda argmo       ARG: M2
03640  4cdf  a4 6e           ldy arglo       ARG: M1
03641  4ce1  20 85 87        jsr fretmp      String ac/yr vom Stapel, ungültig
03642  4ce4  86 6d           stx argmo       ARG: M2
03643  4ce6  84 6e           sty arglo       ARG: M1
03644  4ce8
03645  4ce8  aa              tax
03646  4ce9  38              sec
03647  4cea  e5 63           sbc facexp      FAC: EX
03648  4cec  f0 08           beq stasgn
03649  4cee
03650  4cee  a9 01           lda #1
03651  4cf0  90 04           bcc stasgn
03652  4cf2
03653  4cf2  a6 63           ldx facexp      FAC: EX
03654  4cf4  a9 ff           lda #$ff
03655  4cf6  85 68     stasgn sta facsgn      FAC: SG
```

```
03656  4cf8  a0 ff            ldy #$ff
03657  4cfa  e8               inx
03658  4cfb  c8        nxtcmp iny
03659  4cfc  ca               dex
03660  4cfd  d0 07            bne getcmp
03661  4cff
03662  4cff  a6 68            ldx facsgn       FAC: SG
03663  4d01
03664  4d01
03665  4d01  ===> Wahrheitswert nach fac <===
03666  4d01
03667  4d01  30 1b      qcomp bmi docmp
03668  4d03
03669  4d03  18               clc
03670  4d04  90 18            bcc docmp
03671  4d06
03672  4d06  a9 6d      getcmp lda #<argmo      ARG: M2
03673  4d08  20 ab 03         jsr iram1        LDA (ac),y aus VAR-RAM
03674  4d0b  48               pha
03675  4d0c
03676  4d0c  a9 64            lda #<facho      FAC: M4
03677  4d0e  20 ab 03         jsr iram1        LDA (ac),y aus VAR-RAM
03678  4d11  85 79            sta syntmp
03679  4d13
03680  4d13  68               pla
03681  4d14  c5 79            cmp syntmp
03682  4d16  f0 e3            beq nxtcmp
03683  4d18
03684  4d18  a2 ff            ldx #$ff
03685  4d1a  b0 02            bcs docmp
03686  4d1c
03687  4d1c  a2 01            ldx #1
03688  4d1e  e8         docmp  inx
03689  4d1f  8a               txa
03690  4d20  2a               rol a
03691  4d21  25 14            and tansgn       Flag für TAN-Vorzeichen
03692  4d23  f0 02            beq goflot
03693  4d25
03694  4d25  a9 ff            lda #$ff
03695  4d27  4c 68 8c   goflot jmp float       8 Bit INT ac nach FAC
03696  4d2a
03697  4d2a
03698  4d2a  ===> Ausgabe »ready.« <===
03699  4d2a
03700  4d2a  20 81 92   reddy  jsr cprimm       folg. Text ausg./überspringen
03701  4d2d
03702  4d2d  0d 52 45          .byte _cr, 'ready.', _cr, _eom
03702  4d30  41 44 59
03702  4d33  2e 0d 00
03703  4d36
03704  4d36  60               rts
03705  4d37
03706  4d37
03707  4d37  ===> Ausgabe »ready.« und ready-Status <===
03708  4d37
03709  4d37  a2 80      ready  ldx #$80
03710  4d39  2c               .byte _skip2
03711  4d3a
```

```
03712  4d3a
03713  4d3a  ===> Behandlung »?out of memory error« <===
03714  4d3a
03715  4d3a  a2 10      omerr  ldx #_erom
03716  4d3c
03717  4d3c
03717  4d3c  ===> ind. Behandlung Fehlermeldungen <===
03718  4d3c
03719  4d3c  6c 00 03   error  jmp (ierror)     ind. Fehlerbehandlung ($4d3f)
03720  4d3f
03721  4d3f
03722  4d3f  ===> Behandlung Fehlermeldung <===
03723  4d3f
03724  4d3f  8d 03 ff   nerror sta lcrc         ROM / PGM-RAM einschalten
03725  4d42
03726  4d42  8a                txa              Nummer Fehlertext
03727  4d43  30 72              bmi readyx       Nummer für »ready.«
03728  4d45
03729  4d45  8e 08 12          stx errnum       Nummer Fehlertext
03730  4d48
03731  4d48  24 7f              bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
03732  4d4a  10 30              bpl errisd       Ausgabe Fehlertext, Zeilennummer und
                                                 ready-Status
03733  4d4c
03734  4d4c  ===> PGM-Zeile mit Fehler speichern <===
03735  4d4c
03736  4d4c  a0 01              ldy #1
03737  4d4e  b9 3b 00   erro1   lda curlin,y     Low-Byte akt. Zeilennummer
03738  4d51  99 09 12          sta errlin,y     Low-Byte Zeilennummer für RESUME
03739  4d54
03740  4d54  b9 02 12          lda oldtxt,y     ADDRL Zeiger letzter BASIC-Befehl
03741  4d57  99 0e 12          sta errtxt,y     ADDRL txtptr bei Fehler
03742  4d5a
03743  4d5a  88                dey
03744  4d5b  10 f1              bpl erro1
03745  4d5d
03746  4d5d  ===> Test auf ERROR TRAP <===
03747  4d5d
03748  4d5d  ac 0c 12          ldy trapno+1     high Zeilennr. TRAP Routine
03749  4d60  c8                iny
03750  4d61  f0 19              beq errisd       Ausgabe Fehlertext, Zeilennummer und
                                                 ready-Status
03751  4d63
03752  4d63  88                dey
03753  4d64  84 17              sty linnum+1     High-Byte akt. Zeilennummer
03754  4d66  8c 0d 12          sty tmptrp       high Zeilennr. TRAP Routine bei TRAP
03755  4d69
03756  4d69  ac 0b 12          ldy trapno       low Zeilennr. TRAP Routine
03757  4d6c  84 16              sty linnum       Low-Byte akt. Zeilennummer
03758  4d6e
03759  4d6e  a2 ff              ldx #$ff         Fehler in TRAP Routine führen
03760  4d70  8e 0c 12          stx trapno+1     zu Fehlermeldung, ready
03761  4d73
03762  4d73  a6 82              ldx oldstk       Stapelzeiger für trap
03763  4d75  9a                txs
03764  4d76
03765  4d76  20 fb 59          jsr lukall       txtptr auf TRAP Routine
03766  4d79  4c f6 4a          jmp newstt       TRAP Routine abarbeiten
```

```
03767  4d7c
03768  4d7c
03769  4d7c  ===> Ausgabe Fehlertext, Zeilennummer und ready-Status <===
03770  4d7c
03771  4d7c  ca          errisd dex              Nummer Fehlertext
03772  4d7d  8a                 txa              Rückwärtszähler
03773  4d7e
03774  4d7e  20 82 4a           jsr erstup       index2 auf Fehlertext (Nummer in xr)
                                                 einstellen
03775  4d81  20 6f 92           jsr cclrch       ind. I/O-Kanäle schließen
03776  4d84
03777  4d84  a9 00              lda #_keybd      Kanalnummer Tastatur
03778  4d86  85 15              sta channl       aktiver I/O-Kanal
03779  4d88
03780  4d88  24 d7              bit mode         BS-Modus 40/80-Zeichen
03781  4d8a  30 02              bmi error2       80-Zeichen BS
03782  4d8c
03783  4d8c  85 d8              sta graphm       40-Zeichen BS: Textmodus GRAPH 0
03784  4d8e
03785  4d8e
03786  4d8e  ===> Ausgabe Fehlermeldung BS <===
03787  4d8e
03788  4d8e  20 98 55    error2 jsr crdo         Ausgabe cr (+lf) auf akt. Kanal
03789  4d91  20 0a 56           jsr outqst       Ausgabe Fragez. akt. Kanal
03790  4d94
03791  4d94  a0 00              ldy #0
03792  4d96  b1 26       geterr lda (index2),y  Zeichen aus Fehlermeldung
03793  4d98
03794  4d98  48                 pha              Zeichen
03795  4d99  29 7f              and #%01111111
03796  4d9b  20 0c 56           jsr outdo        Ausgabe ac auf akt. Kanal
03797  4d9e  c8                 iny              Zeiger in Fehlermeldung
03798  4d9f  68                 pla              letztes Zeichen
03799  4da0
03800  4da0  10 f4              bpl geterr       nächstes Zeichen
03801  4da2
03802  4da2  20 38 52           jsr stkini       reset Stapel- u. BASIC-Befehl-Zeiger
03803  4da5  20 81 92           jsr cprimm       folg. Text ausg./überspringen
03804  4da8
03805  4da8  20 45 52           .byte ' error', _eom
03805  4dab  52 4f 52
03805  4dae  00
03806  4daf
03807  4daf
03808  4daf                     .end
03809  4daf                     .lib basic.main
03810  4daf  put"@basic.main"
```

```
03812  4daf
03813  4daf
03814  4daf  ===> Test PGM-Modus, Ausg. Zeilennr. <===
03815  4daf
03816  4daf  a4 3c      errfin ldy curlin+1     High-Byte akt. Zeilennummer
03817  4db1  c8               iny
03818  4db2  f0 03            beq readyx       Direkt-Modus
03819  4db4
03820  4db4  20 26 8e         jsr inprt        Ausgabe »in .. Zeilennummer«
03821  4db7
03822  4db7  20 2a 4d  readyx jsr reddy        Ausgabe »ready.«
03823  4dba
03824  4dba  a9 80             lda #%10000000   OPSYS-Meld. Direkt-Modus zulassen
03825  4dbc  20 90 ff          jsr kstmsg       Flag setzen Ausgabe/Unterdr.
                                                OPSYS-Meldung (Sprungv.)
03826  4dbf
03827  4dbf  a9 00             lda #0
03828  4dc1  85 7f             sta runmod       Direkt-Modus
03829  4dc3
03830  4dc3  ===> ind. BASIC Eingabeschleife <===
03831  4dc3
03832  4dc3  6c 02 03  main    jmp (imain)      ind. BASIC Eingabeschleife ($4dc6)
03833  4dc6
03834  4dc6
03835  4dc6  ===> BASIC Eingabeschleife <===
03836  4dc6
03837  4dc6  a2 ff      nmain  ldx #$ff          Flag für
03838  4dc8  86 3c             stx curlin+1     Direkt-Modus
03839  4dca
03840  4dca  20 93 4f          jsr inlin        Eingabe in Puffer
03841  4dcd  86 3d      execln stx txtptr       ADDRL Eingabepuffer-1
03842  4dcf  84 3e             sty txtptr+1     ADDRH Eingabepuffer-1
03843  4dd1
03844  4dd1  20 80 03          jsr chrget       nächst. Zeich. Eingabepuff. nach ac
03845  4dd4  aa                tax              Flags
03846  4dd5  f0 ec      main00 beq main         Endezeichen, Warteschleife
03847  4dd7
03848  4dd7  90 09             bcc main1        Zeilennummer einer PGM-Zeile
03849  4dd9
03850  4dd9  20 0a 43          jsr crunch       ind. Wandlung Text/Tokens
03851  4ddc  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
03852  4ddf  4c f3 4a          jmp xeqdir       Zeile im Direktmodus ausführen
03853  4de2
03854  4de2
03855  4de2  ===> Bearbeitung einer Programmzeile (an-, einfügen/löschen) <===
03856  4de2
03857  4de2  20 a0 50   main1  jsr linget       Zeilennr. nach linnum
03858  4de5  20 0a 43          jsr crunch       ind. Wandlung Text/Tokens
03859  4de8  84 0d             sty count        Länge neue PGM-Zeile ohne Kopf
03860  4dea
03861  4dea  20 64 50          jsr fndlin       Adr. linnum/+1 nach lowtr/+1
03862  4ded  90 7b             bcc nodel        neue Zeilennr.
03863  4def
03864  4def  ===> Zeile ersetzen <===
03865  4def
03866  4def  a0 00             ldy #0
03867  4df1  20 ec 42          jsr indlo0       Low-Byte Linkadresse
03868  4df4
```

```
03869  4df4
03869  4df4  ===> Länge alte Zeile berechnen <===
03870  4df4
03871  4df4  38              sec
03872  4df5  e5 61           sbc lowtr        ADDRL Linkadresse
03873  4df7  38              sec
03874  4df8  e9 04           sbc #_hedln      Länge PGM-Zeilen Kopf
03875  4dfa  e5 0d           sbc count        Länge neue PGM-Zeile ohne Kopf
03876  4dfc  b0 1c           bcs main30       alte Zeile länger
03877  4dfe
03878  4dfe  ===> neue Zeile länger, PGM-Rest verschieben <===
03879  4dfe
03880  4dfe  49 ff           eor #%11111111
03881  4e00  69 01           adc #%00000001   Längendifferenz
03882  4e02
03883  4e02  ac 11 12        ldy txttop+1     ADDRH Obergrenze akt. PGM
03884  4e05  6d 10 12        adc txttop       ADDRL Obergrenze akt. PGM
03885  4e08  90 01           bcc main10       Test auf genügend PGM-RAM
03886  4e0a
03887  4e0a  c8              iny
03888  4e0b
03889  4e0b
03890  4e0b  ===> Test auf genügend PGM-RAM <===
03891  4e0b
03892  4e0b  cc 13 12   main10 cpy mxmem0+1   ADDRH Obergr. PGM RAM
03893  4e0e  90 0a           bcc main30       Parameter Verschiebeschleife
                                              berechnen
03894  4e10  d0 05           bne omerr5       Behandlung »?out of memory error«
03895  4e12
03896  4e12  cd 12 12        cmp mxmem0       ADDRL Obergr. PGM RAM
03897  4e15  90 03           bcc main30       Parameter Verschiebeschleife
                                              berechnen
03898  4e17
03899  4e17  ===> Behandlung »?out of memory error« <===
03900  4e17
03901  4e17  4c 3a 4d   omerr5 jmp omerr      Behandlung »?out of memory error«
03902  4e1a
03903  4e1a
03904  4e1a  ===> Parameter Verschiebeschleife berechnen <===
03905  4e1a
03906  4e1a  a0 01      main30 ldy #1
03907  4e1c  20 ec 42         jsr indlo0       ADDRH Linkadresse
03908  4e1f  85 25            sta index1+1
03909  4e21
03910  4e21  ad 10 12         lda txttop       ADDRL Obergrenze akt. PGM
03911  4e24  85 24            sta index1
03912  4e26
03913  4e26  a5 62            lda lowtr+1      ADDRH akt. PGM-Zeile
03914  4e28  85 27            sta index2+1
03915  4e2a
03916  4e2a  88               dey
03917  4e2b  20 ec 42         jsr indlo0       ADDRL Linkadresse
03918  4e2e
03919  4e2e  18               clc
03920  4e2f  e5 61            sbc lowtr        ADDRL akt. PGM-Zeile
03921  4e31  49 ff           eor #%11111111
03922  4e33  18               clc
03923  4e34  6d 10 12         adc txttop       ADDRL Obergrenze akt. PGM
```

```
03924  4e37  8d 10 12          sta txttop       ADDRL Obergrenze akt. PGM
03925  4e3a  85 26             sta index2
03926  4e3c
03927  4e3c  ad 11 12          lda txttop+1     ADDRH Obergrenze akt. PGM
03928  4e3f  69 ff             adc #$ff
03929  4e41  8d 11 12          sta txttop+1     ADDRH Obergrenze akt. PGM
03930  4e44
03931  4e44  e5 62             sbc lowtr+1      ADDRH akt. PGM-Zeile
03932  4e46  aa                tax              Anzahl 256-Byte-Blöcke
03933  4e47
03934  4e47  38                sec
03935  4e48  a5 61             lda lowtr
03936  4e4a  ed 10 12          sbc txttop       ADDRL Obergrenze akt. PGM
03937  4e4d  a8                tay              Anzahl Bytes Anfangsblock
03938  4e4e
03939  4e4e  b0 03             bcs qdect1
03940  4e50  e8                inx              Anzahl 256-Byte-Blöcke
03941  4e51  c6 27             dec index2+1
03942  4e53
03943  4e53  18          qdect1 clc
03944  4e54  65 24             adc index1
03945  4e56  90 03             bcc mloop        Verschiebeschleife
03946  4e58  c6 25             dec index1+1
03947  4e5a
03948  4e5a  18                clc
03949  4e5b
03950  4e5b
03951  4e5b  ===> Verschiebeschleife <===
03952  4e5b
03953  4e5b  20 05 43    mloop  jsr ind1_0      LDA (index1),y aus PGM-RAM
03954  4e5e  91 26             sta (index2),y
03955  4e60
03956  4e60  c8                iny              Zeiger in 256-Byte-Block
03957  4e61  d0 f8             bne mloop        Verschiebeschleife
03958  4e63
03959  4e63  e6 25             inc index1+1
03960  4e65  e6 27             inc index2+1
03961  4e67  ca                dex              Anzahl 256-Byte-Blöcke
03962  4e68  d0 f1             bne mloop        Verschiebeschleife
03963  4e6a
03964  4e6a
03965  4e6a  ===> Test leere Zeile mit neuer Zeilennr. <===
03966  4e6a
03967  4e6a  20 38 52    nodel  jsr stkini      reset Stapel- u. BASIC-Befehl-Zeiger
03968  4e6d  20 4f 4f           jsr lnkprg      alle Linkadressen neu berechnen
03969  4e70
03970  4e70  a0 00             ldy #0
03971  4e72  b1 3d             lda (txtptr),y   erstes Zeichen Eingabepuffer
03972  4e74  d0 03             bne nodele       nicht Zeilenende
03973  4e76
03974  4e76  4c d5 4d          jmp main00       Eingabeschleife
03975  4e79
03976  4e79
03977  4e79  ===> Zeile mit neuer Zeilennr. einfügen <===
03978  4e79
03979  4e79  18          nodele clc
03980  4e7a
03981  4e7a  ad 10 12          lda txttop       ADDRL Obergrenze akt. PGM
```

```
03982  4e7d  ac 11 12          ldy txttop+1     ADDRH Obergrenze akt. PGM
03983  4e80
03984  4e80  85 5c             sta hightr
03985  4e82  84 5d             sty hightr+1
03986  4e84
03987  4e84  65 0d             adc count        Länge neue PGM-Zeile ohne Kopf
03988  4e86  90 01             bcc nodel1
03989  4e88  c8                iny
03990  4e89  18          nodel1 clc
03991  4e8a  69 04             adc #_hedln      Länge PGM-Zeilen-Kopf
03992  4e8c  90 01             bcc nodelc
03993  4e8e  c8                iny
03994  4e8f  85 5a       nodelc sta highds
03995  4e91  84 5b             sty highds+1
03996  4e93
03997  4e93  cc 13 12          cpy mxmem0+1     ADDRH Obergr. PGM RAM
03998  4e96  90 0a             bcc nodel2       Parameter Verschiebeschleife
                                                berechnen
03999  4e98
04000  4e98  d0 05             bne omerr2       Behandlung »?out of memory error«
04001  4e9a  cd 12 12          cmp mxmem0       ADDRL Obergr. PGM RAM
04002  4e9d  90 03             bcc nodel2       Parameter Verschiebeschleife
                                                berechnen
04003  4e9f
04004  4e9f  ===> Behandlung »?out of memory error« <===
04005  4e9f
04006  4e9f  4c 3a 4d    omerr2 jmp omerr        Behandlung »?out of memory error«
04007  4ea2
04008  4ea2
04009  4ea2  ===> Parameter Verschiebeschleife berechnen <===
04010  4ea2
04011  4ea2  8d 10 12    nodel2 sta txttop       ADDRL Obergrenze akt. PGM
04012  4ea5  8c 11 12           sty txttop+1     ADDRH Obergrenze akt. PGM
04013  4ea8
04014  4ea8  38                sec
04015  4ea9  a5 5c             lda hightr
04016  4eab  e5 61             sbc lowtr
04017  4ead  85 24             sta index1
04018  4eaf
04019  4eaf  a8                tay              Anzahl Bytes Anfangsblock
04020  4eb0  a5 5d             lda hightr+1
04021  4eb2  e5 62             sbc lowtr+1
04022  4eb4
04023  4eb4  aa                tax              Anzahl 256-Byte-Blöcke
04024  4eb5  e8                inx
04025  4eb6
04026  4eb6  98                tya              Anzahl Bytes Anfangsblock
04027  4eb7  f0 25             beq nodel6
04028  4eb9
04029  4eb9  a5 5c             lda hightr
04030  4ebb  38                sec
04031  4ebc  e5 24             sbc index1
04032  4ebe  85 5c             sta hightr
04033  4ec0  b0 03             bcs nodel3
04034  4ec2  c6 5d             dec hightr+1
04035  4ec4
04036  4ec4  38                sec
04037  4ec5  a5 5a       nodel3 lda highds
```

```
04038  4ec7  e5 24           sbc index1
04039  4ec9  85 5a           sta highds
04040  4ecb  b0 09           bcs nodel5
04041  4ecd  c6 5b           dec highds+1
04042  4ecf  90 05           bcc nodel5
04043  4ed1
04044  4ed1
04045  4ed1  ===> Verschiebeschleife <===
04046  4ed1
04047  4ed1  20 dd 42  nloop  jsr inhtr0      LDA (hightr),y aus PGM-RAM
04048  4ed4  91 5a           sta (highds),y
04049  4ed6
04050  4ed6  88        nodel5 dey             Zeiger in 256-Byte-Block
04051  4ed7  d0 f8           bne nloop        Verschiebeschleife
04052  4ed9
04053  4ed9  20 dd 42        jsr inhtr0       LDA (hightr),y aus PGM-RAM
04054  4edc  91 5a           sta (highds),y
04055  4ede
04056  4ede  c6 5d     nodel6 dec hightr+1
04057  4ee0  c6 5b           dec highds+1
04058  4ee2  ca              dex              Anzahl 256-Byte-Blöcke
04059  4ee3  d0 f1           bne nodel5
04060  4ee5
04061  4ee5  ===> Linkadresse lnkdmy ($0101) in Zeilenkopf <===
04062  4ee5
04063  4ee5  a0 00           ldy #0
04064  4ee7  a9 01           lda #<_lnkdy     Dummy Linkadresse $0101
04065  4ee9  91 61           sta (lowtr),y    ADDRL Linkadresse
04066  4eeb
04067  4eeb  c8              iny
04068  4eec  91 61           sta (lowtr),y    ADDRH Linkadresse
04069  4eee
04070  4eee  ===> neue Zeilennummer in Zeilenkopf <===
04071  4eee
04072  4eee  c8              iny
04073  4eef  a5 16           lda linnum       Low-Byte akt. Zeilennummer
04074  4ef1  91 61           sta (lowtr),y
04075  4ef3
04076  4ef3  a5 17           lda linnum+1     High-Byte akt. Zeilennummer
04077  4ef5  c8              iny
04078  4ef6  91 61           sta (lowtr),y
04079  4ef8
04080  4ef8  ===> Zeiger lowtr hinter Zeilenkopf einstellen <===
04081  4ef8
04082  4ef8  18              clc
04083  4ef9  a5 61           lda lowtr
04084  4efb  69 04           adc #_hedln      Länge PGM-Zeilen-Kopf
04085  4efd  85 61           sta lowtr
04086  4eff  90 02           bcc main2
04087  4f01  e6 62           inc lowtr+1
04088  4f03
04089  4f03  a4 0d     main2 ldy count        Länge neue PGM-Zeile ohne Kopf
04090  4f05  88              dey              Zeiger in Eingabepuffer
04091  4f06
04092  4f06
04093  4f06  ===> Zeile aus Eingabepuffer nach PGM-RAM <===
04094  4f06
04095  4f06  b1 3d     stolop lda (txtptr),y  Zeichen aus Eingabepuffer
```

```
04096  4f08  91 61          sta (lowtr),y    ins PGM-RAM
04097  4f0a  88             dey              Zeiger in Eingabepuffer
04098  4f0b  c0 ff          cpy #$ff
04099  4f0d  d0 f7          bne stolop
04100  4f0f
04101  4f0f  20 4f 4f       jsr lnkprg       alle Linkadressen neu berechnen
04102  4f12  20 54 52       jsr stxtpt       txtptr auf PGM Anf.
04103  4f15
04104  4f15  ===> Test auf AUTO-Number <===
04105  4f15
04106  4f15  a5 74          lda autinc       Schrittweite low für AUTO
04107  4f17  05 75          ora autinc+1     Schrittweite high für AUTO
04108  4f19  f0 31          beq gomain       AUTO-Number aus
04109  4f1b
04110  4f1b  ===> nächste Zeilennr. berechnen / vorlegen <===
04111  4f1b
04112  4f1b  a5 16          lda linnum       Low-Byte akt. Zeilennummer
04113  4f1d  18             clc
04114  4f1e  65 74          adc autinc       Schrittweite low für AUTO
04115  4f20  85 65          sta facmoh       FAC: M3
04116  4f22
04117  4f22  a5 17          lda linnum+1     High-Byte akt. Zeilennummer
04118  4f24  65 75          adc autinc+1     Schrittweite high für AUTO
04119  4f26  b0 24          bcs gomain       nächste Zeilennr. größer 65535
04120  4f28
04121  4f28  c9 fa          cmp #>64000
04122  4f2a  b0 20          bcs gomain       nächste Zeilennr. größer 63999
04123  4f2c
04124  4f2c  85 64          sta facho        FAC: M4
04125  4f2e
04126  4f2e  a2 90          ldx #_expcn+15
04127  4f30  38             sec
04128  4f31  20 75 8c       jsr floatc       EX nach FAC, Nullen auffüllen,
                                             normal.
04129  4f34  20 42 8e       jsr fout         FAC nach String, Adr. nach ac/yr
04130  4f37
04131  4f37  ===> Vorgabe nächste Zeilennr. nach Tast. puffer <===
04132  4f37
04133  4f37  a2 00          ldx #0
04134  4f39  bd 01 01  aline1 lda fbuffr+1,x  Zeichen aus String neue Zeilennr.
04135  4f3c  f0 06          beq aline2       Endezeichen
04136  4f3e
04137  4f3e  9d 4a 03       sta keyd,x       Tast.puffer
04138  4f41  e8             inx
04139  4f42  d0 f5          bne aline1
04140  4f44
04141  4f44  a9 1d     aline2 lda #_crite    Cursor rechts
04142  4f46  9d 4a 03       sta keyd,x       Tast.puffer
04143  4f49  e8             inx
04144  4f4a  86 d0          stx ndx          Bytes im Tast.puffer
04145  4f4c  4c c3 4d  gomain jmp main       ind. BASIC Eingabeschleife
04146  4f4f
04147  4f4f
04148  4f4f  ===> alle Linkadressen neu berechnen <===
04149  4f4f
04150  4f4f  a5 2d      lnkprg lda txttab    ADDRL Untergrenze akt. PGM
04151  4f51  a4 2e             ldy txttab+1  ADDRH Untergrenze akt. PGM
04152  4f53
```

```
04153  4f53  85 24            sta index1
04154  4f55  84 25            sty index1+1
04155  4f57
04156  4f57  18               clc
04157  4f58
04158  4f58
04159  4f58  ===> Test auf Programm in PGM-RAM <===
04160  4f58
04161  4f58  a0 00      chead  ldy #0
04162  4f5a  20 05 43          jsr ind1_0       Low-Byte Linkadresse
04163  4f5d  d0 06             bne chea3
04164  4f5f
04165  4f5f  c8                iny
04166  4f60  20 05 43          jsr ind1_0       High-Byte Linkadresse
04167  4f63  f0 2d             beq lnkrts       kein Programm im PGM-RAM
04168  4f65
04169  4f65  a0 04      chea3  ldy #_hedln      Zeiger hinter Zeilenkopf
04170  4f67  c8         czloop iny              Zeiger in PGM-Zeile
04171  4f68  20 05 43          jsr ind1_0       Zeichen aus PGM-Zeile
04172  4f6b  d0 fa             bne czloop
04173  4f6d
04174  4f6d  ===> Endezeichen PGM-Zeile gefunden <===
04175  4f6d
04176  4f6d  c8                iny              Länge PGM-Zeile
04177  4f6e  98                tya
04178  4f6f  65 24             adc index1       ADDRL Zeilenkopf akt. Zeile
04179  4f71  aa                tax              Low-Byte Linkadresse akt. Zeile
04180  4f72
04181  4f72  a0 00             ldy #0
04182  4f74  91 24             sta (index1),y   Low-Byte Linkadr. akt. Zeile
04183  4f76  98                tya              0
04184  4f77  65 25             adc index1+1     ADDRH Zeilenkopf akt. Zeile
04185  4f79
04186  4f79  c8                iny
04187  4f7a  91 24             sta (index1),y   High-Byte Linkadr. akt. Zeile
04188  4f7c
04189  4f7c  ===> Zeiger auf nächste PGM-Zeile <===
04190  4f7c
04191  4f7c  86 24             stx index1
04192  4f7e  85 25             sta index1+1
04193  4f80  90 d6             bcc chead        nächste Linkadresse berechnen/
                                                eintragen
04194  4f82
04195  4f82
04196  4f82  ===> Obergrenze akt. PGM nachführen <===
04197  4f82
04198  4f82  18         fixlnk clc
04199  4f83  a5 24              lda index1
04200  4f85  a4 25              ldy index1+1
04201  4f87  69 02              adc #2
04202  4f89  90 01              bcc lnkrt1
04203  4f8b  c8                 iny
04204  4f8c  8d 10 12   lnkrt1  sta txttop       ADDRL Obergrenze akt. PGM
04205  4f8f  8c 11 12           sty txttop+1     ADDRH Obergrenze akt. PGM
04206  4f92  60         lnkrts  rts
04207  4f93
04208  4f93
```

```
04209  4f93  ===> Eingabe in Puffer <===
04210  4f93
04211  4f93  a2 00      inlin  ldx #0           Zeiger in Eingabepuffer
04212  4f95
04213  4f95  20 e5 90   inlinc jsr inchr        Zeichen einlesen
04214  4f98  c9 0d             cmp #_cr         Carriage Return
04215  4f9a  d0 03             bne inlind
04216  4f9c
04217  4f9c  4c 8b 55          jmp fininl       Abschluß Zeilen-Eingabe
04218  4f9f
04219  4f9f
04220  4f9f  9d 00 02   inlind sta buf,x        Zeichen in Eingabepuffer
04221  4fa2  e8                inx              Zeiger in Eingabepuffer
04222  4fa3
04223  4fa3  e0 a1             cpx #_bufln      161 Zeichen
04224  4fa5  90 ee             bcc inlinc
04225  4fa7
04226  4fa7  ===> Behandlung »?string too long error« <===
04227  4fa7
04228  4fa7  4c ed a5          jmp errlen       Behandlung »?string too long«
04229  4faa
04230  4faa
04231  4faa  ===> Token (ac) in BASIC-Stapel suchen <===
04232  4faa
04233  4faa  85 02      search sta srchtk        Such Token
04234  4fac  20 47 50          jsr movtos       Kopie Stapelzeiger nach Suchzeiger
04235  4faf
04236  4faf
04237  4faf  ===> Test auf Ende BASIC-Stapel <===
04238  4faf
04239  4faf  a5 3f      srch05 lda fndpnt        ADDRL BASIC-Stapel Suchzeiger
04240  4fb1  c9 ff             cmp #<stkbot     Untergrenze BASIC-Stapel
04241  4fb3  d0 06             bne srch10
04242  4fb5
04243  4fb5  a5 40             lda fndpnt+1     ADDRH BASIC-Stapel Suchzeiger
04244  4fb7  c9 09             cmp #>stkbot     Untergrenze BASIC-Stapel
04245  4fb9  f0 40             beq srch98       Ende BASIC-Stapel erreicht
04246  4fbb
04247  4fbb  8d 03 ff   srch10 sta lcrc         ROM / PGM-RAM einschalten
04248  4fbe  a0 00             ldy #0
04249  4fc0  a5 02             lda srchtk       Such Token
04250  4fc2
04251  4fc2  c9 81             cmp #_fort       FOR Token
04252  4fc4  d0 1b             bne srch20
04253  4fc6
04254  4fc6  ===> Suche nach FOR Token im BASIC-Stapel <===
04255  4fc6
04256  4fc6  d1 3f             cmp (fndpnt),y   Token mit Stapel vergleichen
04257  4fc8  d0 33             bne srch99       nicht gefunden, Abbruch
04258  4fca
04259  4fca  a0 02             ldy #2           ADDRH Param. FOR/NEXT Vari. im
                                                Stapel
04260  4fcc  a5 4c             lda lstpnt+1
04261  4fce
04262  4fce  c9 ff             cmp #$ff         Flag für next ohne Variablennamen
04263  4fd0  f0 2b             beq srch99
04264  4fd2
```

```
04265  4fd2  ===> next mit Variablennamen <===
04266  4fd2
04267  4fd2  d1 3f           cmp (fndpnt),y   Vergleich mit Parameter auf Stapel
04268  4fd4  d0 07           bne srch15       Parameter von falscher Variablen
04269  4fd6
04270  4fd6  88              dey              ADDRL Param. FOR/NEXT Vari. im
                                              Stapel
04271  4fd7  a5 4b           lda lstpnt
04272  4fd9  d1 3f           cmp (fndpnt),y   ADDRL BASIC-Stapel Suchzeiger
04273  4fdb  f0 20           beq srch99       gefunden
04274  4fdd
04275  4fdd  a2 12     srch15 ldx #_lnfor     Stapel-Bytes für FOR/NEXT Schleife
04276  4fdf  d0 0e           bne srch30       immer
04277  4fe1
04278  4fe1
04279  4fe1  b1 3f     srch20 lda (fndpnt),y  ADDRL BASIC-Stapel Suchzeiger
04280  4fe3  c5 02           cmp srchtk       Such Token
04281  4fe5  f0 16           beq srch99
04282  4fe7
04283  4fe7  a2 12           ldx #_lnfor      Stapel-Bytes für FOR/NEXT Schleife
04284  4fe9  c9 81           cmp #_fort       FOR Token
04285  4feb  f0 02           beq srch30       Stapelzeiger erhöhen
04286  4fed
04287  4fed  a2 05           ldx #_lngos      Stapel-Byte für GOSUB
04288  4fef
04289  4fef
04290  4fef  ===> Stapelzeiger erhöhen <===
04291  4fef
04292  4fef  8a       srch30 txa              Anzahl Stapel-Bytes
04293  4ff0  18              clc
04294  4ff1  65 3f           adc fndpnt       ADDRL BASIC-Stapel Suchzeiger
04295  4ff3  85 3f           sta fndpnt       ADDRL BASIC-Stapel Suchzeiger
04296  4ff5  90 b8           bcc srch05       Test auf Ende BASIC-Stapel
04297  4ff7  e6 40           inc fndpnt+1     ADDRH BASIC-Stapel Suchzeiger
04298  4ff9  d0 b4           bne srch05       Test auf Ende BASIC-Stapel
04299  4ffb
04300  4ffb  a0 01     srch98 ldy #1          RTS mit ZERO-Flag = 0
04301  4ffd  60        srch99 rts
04302  4ffe
04303  4ffe
04304  4ffe  ===> prüfen auf Platz im Stapel <===
04305  4ffe
04306  4ffe  49 ff     getstk eor #$ff
04307  5000  38              sec
04308  5001  65 7d           adc tos          ADDRL BASIC-Stapel Zeiger
04309  5003  85 7d           sta tos          ADDRL BASIC-Stapel Zeiger
04310  5005
04311  5005  a4 7e           ldy tos+1        ADDRH BASIC-Stapel Zeiger
04312  5007  b0 01           bcs getst1
04313  5009  88              dey
04314  500a  84 7e     getst1 sty tos+1        ADDRH BASIC-Stapel Zeiger
04315  500c  c0 08           cpy #>stktop     Obergrenze BASIC-Stapel
04316  500e  90 34           bcc omerr6       Behandlung »?out of memory error«
04317  5010  d0 04           bne getsrt
04318  5012
04319  5012  c5 7d           cmp tos          ADDRL BASIC-Stapel Zeiger
04320  5014  90 2e           bcc omerr6       Behandlung »?out of memory error«
04321  5016
```

```
04322  5016  60         getsrt rts
04323  5017
04324  5017
04325  5017  ===> Test Platz in VAR-RAM, sonst garb.coll. <===
04326  5017
04327  5017  c4 36      reason cpy fretop+1     ADDRH Zeiger Anfang String-Variable
04328  5019  90 28             bcc reasrt       genügend Speicherplatz
04329  501b
04330  501b  d0 04             bne reas10       Register/Zeiger retten, garb.coll.
04331  501d
04332  501d  c5 35             cmp fretop       ADDRL Zeiger Anfang String-Variable
04333  501f  90 22             bcc reasrt       genügend Speicherplatz
04334  5021
04335  5021
04336  5021  ===> Register/Zeiger retten, garb.coll. <===
04337  5021
04338  5021  48         reas10 pha              ac auf Stapel
04339  5022
04340  5022  a2 09             ldx #lowtr+1-tempf1    TMP FAC1
04341  5024
04342  5024  98                tya              yr auf Stapel
04343  5025
04344  5025  48         reas20 pha
04345  5026  b5 59             lda tempf1,x
04346  5028  ca                dex
04347  5029  10 fa             bpl reas20
04348  502b
04349  502b  20 ea 92          jsr garba2       Garbage Collection
04350  502e
04351  502e  a2 f7             ldx #256-lowtr-1+tempf1
04352  5030
04353  5030
04354  5030  ===> tempf1 bis lowtr+1 vom Stapel <===
04355  5030
04356  5030  68         reas30 pla
04357  5031  95 63             sta lowtr+2,x
04358  5033  e8                inx
04359  5034  30 fa             bmi reas30       tempf1 bis lowtr+1 vom Stapel
04360  5036
04361  5036  68                pla              yr vom
04362  5037  a8                tay              Stapel
04363  5038
04364  5038  68                pla              ac vom Stapel
04365  5039
04366  5039  c4 36             cpy fretop+1     ADDRH Zeiger Anfang String-Variable
04367  503b  90 06             bcc reasrt       jetzt genügend Speicherplatz
04368  503d  d0 05             bne omerr6       Behandlung »?out of memory error«
04369  503f
04370  503f  c5 35             cmp fretop       ADDRL Zeiger Anfang String-Variable
04371  5041  b0 01             bcs omerr6       Behandlung »?out of memory error«
04372  5043
04373  5043  60         reasrt rts
04374  5044
04375  5044
04376  5044  ===> Behandlung »?out of memory error« <===
04377  5044
04378  5044  4c 3a 4d   omerr6 jmp omerr        Behandlung »?out of memory error«
04379  5047
```

```
04380  5047
04381  5047  ===> Kopie Stapelzeiger nach Suchzeiger <===
04382  5047
04383  5047  a5 7d      movtos lda tos          ADDRL BASIC-Stapel Zeiger
04384  5049  85 3f             sta fndpnt       ADDRL BASIC-Stapel Suchzeiger
04385  504b
04386  504b  a5 7e             lda tos+1        ADDRH BASIC-Stapel Zeiger
04387  504d  85 40             sta fndpnt+1     ADDRH BASIC-Stapel Suchzeiger
04388  504f
04389  504f  60               rts
04390  5050
04391  5050
04392  5050  ===> Kopie Suchzeiger nach Stapelzeiger <===
04393  5050
04394  5050  a5 3f      movfnd lda fndpnt       ADDRL BASIC-Stapel Suchzeiger
04395  5052  85 7d             sta tos          ADDRL BASIC-Stapel Zeiger
04396  5054
04397  5054  a5 40             lda fndpnt+1     ADDRH BASIC-Stapel Suchzeiger
04398  5056  85 7e             sta tos+1        ADDRH BASIC-Stapel Zeiger
04399  5058
04400  5058  60               rts
04401  5059
04402  5059
04403  5059  ===> BASIC-Stapel Zeiger um yr erhöhen <===
04404  5059
04405  5059  98         rlsstk tya
04406  505a  18                clc
04407  505b  65 7d             adc tos          ADDRL BASIC-Stapel Zeiger
04408  505d  85 7d             sta tos          ADDRL BASIC-Stapel Zeiger
04409  505f
04410  505f  90 02             bcc rlssrt
04411  5061  e6 7e             inc tos+1        ADDRH BASIC-Stapel Zeiger
04412  5063  60         rlssrt rts
04413  5064
04414  5064
04415  5064  ===> Adr. linnum/+1 nach lowtr/+1 <===
04416  5064
04417  5064  a5 2d      fndlin lda txttab       ADDRL Untergrenze akt. PGM
04418  5066  a6 2e             ldx txttab+1     ADDRH Untergrenze akt. PGM
04419  5068
04420  5068
04421  5068  ===> Adr. linnum nach lowtr, suchen ab ac/yr <===
04422  5068
04423  5068  a0 01      fndlnc ldy #1           Zeiger auf ADDRH Linkadresse
04424  506a
04425  506a  85 61             sta lowtr
04426  506c  86 62             stx lowtr+1
04427  506e
04428  506e  20 ec 42          jsr indlo0       ADDRH Linkadresse nach ac
04429  5071  f0 2b             beq flinrt       PGM-Ende gefunden
04430  5073
04431  5073  c8               iny
04432  5074  c8               iny              Zeiger auf High-Byte Zeilennummer
04433  5075  20 ec 42          jsr indlo0       High-Byte Zeilennr. nach ac
04434  5078  85 79             sta syntmp       High-Byte Zeilennummer
04435  507a
04436  507a  a5 17             lda linnum+1     High-Byte gesuchte Zeilennr.
04437  507c  c5 79             cmp syntmp       mit akt. vergleichen
```

```
04438  507e  90 1f           bcc flnrts      kleiner, nicht gefunden
04439  5080  f0 03           beq fndln3      gleich, gefunden
04440  5082
04441  5082  88              dey             Zeiger auf Low-Byte nächstgrößere
                                             Zeilennr.
04442  5083  d0 0e           bne affrts      immer
04443  5085
04444  5085
04445  5085  88        fndln3 dey             Zeiger auf Low-Byte nächstgrößere
                                             Zeilennr.
04446  5086  20 ec 42        jsr indlo0      Low-Byte akt. Zeilennr. nach ac
04447  5089  85 79          sta syntmp      Low-Byte akt. Zeilennr.
04448  508b
04449  508b  a5 16           lda linnum      Low-Byte gesuchte Zeilennr.
04450  508d  c5 79           cmp syntmp      vergleichen mit Low-Byte akt.
04451  508f
04452  508f  90 0e           bcc flnrts      kleiner, nicht gefunden
04453  5091  f0 0c           beq flnrts      gleich, gefunden
04454  5093
04455  5093
04456  5093  ===> gesuchte Zeilennr. größer, weitersuchen <===
04457  5093
04458  5093  88        affrts dey             Zeiger auf ADDRH Linkadresse
04459  5094  20 ec 42        jsr indlo0      ADDRH Linkadresse
04460  5097  aa              tax             nach xr
04461  5098
04462  5098  88              dey             Zeiger auf ADDRL Linkadresse
04463  5099  20 ec 42        jsr indlo0      ADDRL Linkadresse nach ac
04464  509c  b0 ca           bcs fndlnc      immer
04465  509e
04466  509e
04467  509e  ===> Zeilennr. nicht gefunden <===
04468  509e
04469  509e  18        flinrt clc
04470  509f  60        flnrts rts
04471  50a0
04472  50a0
04473  50a0  ===> Zeilennr. nach linnum <===
04474  50a0
04475  50a0  a2 00     linget ldx #0
04476  50a2  86 0a            stx numflg      Flag Zeilennr. nicht gelesen
04477  50a4  86 16            stx linnum      Low-Byte und
04478  50a6  86 17            stx linnum+1    High-Byte Zeilennr. rücksetzen
04479  50a8
04480  50a8
04481  50a8  ===> nächste Dezimalstelle berechnen <===
04482  50a8
04483  50a8  b0 37     morlin bcs lingrt      CARRY nach chrget: keine Ziffer mehr
04484  50aa
04485  50aa  ===> Wandlung ASCII 0-9 nach HEX <===
04486  50aa
04487  50aa  e6 0a            inc numflg      Flag Zeilennr. gelesen
04488  50ac  e9 2f            sbc #$30-1      (CARRY = 0)
04489  50ae  85 09            sta charac      HEX Nibble
04490  50b0
04491  50b0  a5 17            lda linnum+1    High-Byte bisheriger HEX-Wert
04492  50b2  85 24            sta index1
04493  50b4
```

```
04494  50b4  c9 19           cmp #>64000/10
04495  50b6  90 03           bcc lingt1
04496  50b8
04497  50b8  ===> Behandlung »?syntax error« <===
04498  50b8
04499  50b8  4c 6c 79        jmp snerr        Behandlung »?syntax error«
04500  50bb
04501  50bb
04502  50bb  a5 16     lingt1 lda linnum       16-Bit HEX-Wert
04503  50bd  0a               asl a
04504  50be  26 24            rol index1
04505  50c0  0a               asl a
04506  50c1  26 24            rol index1       mal vier
04507  50c3
04508  50c3  65 16            adc linnum       Low-Byte
04509  50c5  85 16            sta linnum       plus einmal
04510  50c7
04511  50c7  a5 24            lda index1       High-Byte
04512  50c9  65 17            adc linnum+1     plus
04513  50cb  85 17            sta linnum+1     einmal
04514  50cd
04515  50cd  06 16            asl linnum       16-Bit HEX-Wert (bisher mal fünf)
04516  50cf  26 17            rol linnum+1     noch mal zwei -> bisheriger HEX-Wert
                                                mal 10
04517  50d1
04518  50d1  a5 16            lda linnum       zum Zehnfachen des alten HEX-Wertes
04519  50d3  65 09            adc charac       die aktuelle Ziffer
04520  50d5  85 16            sta linnum       addieren
04521  50d7
04522  50d7  90 02            bcc lingt2
04523  50d9  e6 17            inc linnum+1     einschl. Übertrag
04524  50db
04525  50db  20 80 03  lingt2 jsr chrget       wenn nächste Ziffer nach ac: CARRY=0
04526  50de  4c a8 50         jmp morlin       nächste Dezimalstelle berechnen
04527  50e1
04528  50e1
04529  50e1  60        lingrt rts
04530  50e2
04531  50e2
04532  50e2                   .end
04533  50e2                   .lib basic.cmd_1
04534  50e2  put "@basic.cmd_1"
```

```
04536  50e2
04537  50e2
04538  50e2  ===> BASIC-Statement LIST ($9b) <===
04539  50e2
04540  50e2  20 fb 5e   list   jsr range        Zeilenbereich einlesen
04541  50e5
04542  50e5
04543  50e5  ===> Test BASIC-Textende erreicht <===
04544  50e5
04545  50e5  a0 01      list4  ldy #1           Zeiger auf ADDRH Linkadresse
04546  50e7  20 ec 42          jsr indlo0       ADDRH Linkadresse nach ac
04547  50ea  d0 06             bne list44       nicht PGM-Ende
04548  50ec
04549  50ec  88                dey              Zeiger auf ADDRL Linkadresse
04550  50ed  20 ec 42          jsr indlo0       ADDRL Linkadresse nach ac
04551  50f0  f0 2e             beq gordy2       PGM-Ende
04552  50f2
04553  50f2
04554  50f2  ===> Stopp-Taste abfragen/ausführen <===
04555  50f2
04556  50f2  20 b5 4b   list44 jsr istpky       Stopp-Taste abfragen
04557  50f5  20 98 55          jsr crdo         Ausgabe cr (+lf) auf akt. Kanal
04558  50f8
04559  50f8  a0 02             ldy #2           Zeiger auf Low-Byte akt. Zeilennr.
04560  50fa  20 ec 42          jsr indlo0       Low-Byte nach ac
04561  50fd  aa                tax              Low-Byte akt. Zeilennr.
04562  50fe
04563  50fe  c8                iny              Zeiger auf High-Byte akt. Zeilennr.
04564  50ff  20 ec 42          jsr indlo0       High-Byte nach ac
04565  5102
04566  5102  c5 17             cmp linnum+1     High-Byte End-Zeilennr.
04567  5104  d0 04             bne tstdun       prüfen auf Ende list-Bereich
04568  5106
04569  5106  e4 16             cpx linnum       Low-Byte End-Zeilennr.
04570  5108  f0 02             beq typlin       letzte Zeile listen
04571  510a
04572  510a
04573  510a  ===> prüfen auf Ende list-Bereich <===
04574  510a
04575  510a  b0 14      tstdun bcs gordy2       fertig, größer als End-Zeilennr.
04576  510c
04577  510c
04578  510c  ===> Ausgabe Zeilennummer und Space <===
04579  510c
04580  510c  20 23 51   typlin jsr p1line       LIST akt. Zeile
04581  510f
04582  510f  ===> Adresse nächste Zeile nach lowtr/+1 <===
04583  510f
04584  510f  a0 00             ldy #0           Zeiger auf ADDRL Linkadresse
04585  5111  20 ec 42          jsr indlo0       ADDRL nach ac
04586  5114  aa                tax              ADDRL nächste Zeile
04587  5115
04588  5115  c8                iny              Zeiger auf ADDRH
04589  5116  20 ec 42          jsr indlo0       ADDRH Linkadresse nach ac
04590  5119
04591  5119  86 61             stx lowtr        ADDRL nächste Zeile
04592  511b  85 62             sta lowtr+1      ADDRH nächste Zeile
04593  511d
```

```
04594  511d  4c e5 50          jmp list4        Test / weiter listen
04595  5120
04596  5120
04597  5120  4c 98 55   gordy2 jmp crdo         Ausgabe cr (+lf) auf akt. Kanal
04598  5123
04599  5123
04600  5123  ===> LIST akt. Zeile <===
04601  5123
04602  5123  a0 03      p1line ldy #3           Zeiger auf High-Byte Zeilennr.
04603  5125  84 4b             sty lstpnt       Zeiger in Zeile
04604  5127
04605  5127  84 11             sty dores        Flag für Text in Anführungszeichen
04606  5129
04607  5129    7.......  1 = zwischen Anführungszeichen, 0 = außerhalb
04608  5129
04609  5129  20 32 8e           jsr linprt       Ausgabe Zeilennr. (ac/xr) dez. ASCII
04610  512c
04611  512c  a9 20             lda #_spc        Space hinter Zeilennummer
04612  512e
04613  512e  a4 4b      p1l010 ldy lstpnt       Zeiger in Zeile
04614  5130  29 7f             and #%01111111   Bit7 ac löschen
04615  5132
04616  5132
04617  5132  ===> Zeichen aus Zeile in ASC ausgeben, weiter <===
04618  5132
04619  5132  20 0c 56   p1l015 jsr outdo        Ausgabe ac auf akt. Kanal
04620  5135
04621  5135  ===> Test Anführungszeichen, Flag flippen <===
04622  5135
04623  5135  c9 22             cmp #'"'
04624  5137  d0 06             bne p1l019       Test Zeilenende, dann Zeiger auf
                                                nächste Zeile
04625  5139
04626  5139  a5 11             lda dores        Flag Anführungszeichen
04627  513b  49 ff             eor #%11111111   flippen
04628  513d
04629  513d    7.......  1 = zwischen Anführungszeichen, 0 = außerhalb
04630  513d
04631  513d  85 11             sta dores
04632  513f
04633  513f
04634  513f  ===> Test Zeilenende, dann Zeiger auf nächste Zeile <===
04635  513f
04636  513f  c8          p1l019 iny              Zeiger in akt. Zeile
04637  5140  f0 de             beq gordy2       256 Zeichen
04638  5142
04639  5142  24 55             bit helper       Flag HELP / LIST
04640  5144
04641  5144    7.......  1 = HELP, 0 = LIST
04642  5144
04643  5144  10 03             bpl p1l020       normales LIST
04644  5146
04645  5146  20 ac 59           jsr helpsb       Test Fehlerspalte, RVS/BLINK ON
04646  5149
04647  5149  20 ec 42   p1l020 jsr indlo0       nächstes Zeichen aus PGM-Zeile
04648  514c  f0 3d             beq p1lrts       Endezeichen
04649  514e
```

```
04650  514e  6c 06 03          jmp (iqplop)      ind. LIST/HELP Wandlung Tokens/Text
                                                 ($5151)
04651  5151
04652  5151
04653  5151  ===> LIST/HELP Wandlung Tokens/Text <===
04654  5151
04655  5151  10 df       nqplop bpl p1l015       Ausgabe in ASCII (kein Token)
04656  5153
04657  5153  c9 ff              cmp #_pi
04658  5155  f0 db             beq p1l015        Ausgabe »pi«
04659  5157
04660  5157  24 11             bit dores         Flag Anführungszeichen
04661  5159
04662  5159    7.......  1 = zwischen Anführungszeichen, 0 = außerhalb
04663  5159
04664  5159  30 d7             bmi p1l015        Großbuchstabe in Anführungszeichen
04665  515b
04666  515b
04667  515b  ===> Token, Wandlung in Text <===
04668  515b
04669  515b  c9 fe             cmp #_ecmdt       ESC COMMAND Token
04670  515d  f0 2d             beq prescc        Wandlung ESC COMMAND Token/Text
04671  515f
04672  515f  c9 ce             cmp #_efunt       ESC FUNCTION Token
04673  5161  f0 43             beq prescf        Wandlung ESC FUNCTION Token/Text
04674  5163
04675  5163  aa                tax               Token
04676  5164  84 4b             sty lstpnt        Zeiger in akt. Zeile
04677  5166
04678  5166  a9 44             lda #>reslst      BASIC Befehlswort Tabelle
04679  5168  a0 17             ldy #<reslst      BASIC Befehlswort Tabelle
04680  516a
04681  516a
04682  516a  ===> Ausg. Text Befehlswort-Tab. ac/yr <===
04683  516a
04684  516a  85 25       p1l026 sta index1+1      ADDRH Zeiger Befehlswort-Tab.
04685  516c  84 24              sty index1        ADDRL Zeiger Befehlswort-Tab.
04686  516e
04687  516e  a0 00              ldy #0            Offset in Befehlswort-Tab.
04688  5170  ca         p1l030 dex               Token
04689  5171  10 0f             bpl p1l070        Ausgabe Text zu Token
04690  5173
04691  5173
04692  5173  ===> Text Befehlswort-Tab. überlesen <===
04693  5173
04694  5173  b1 24       p1l040 lda (index1),y
04695  5175  48                 pha               Zeichen aus Befehlswort-Tab.
04696  5176  e6 24              inc index1        ADDRL Zeiger Befehlswort-Tab.
04697  5178  d0 02              bne p1l050
04698  517a  e6 25              inc index1+1      ADDRH Zeiger Befehlswort-Tab.
04699  517c
04700  517c  68         p1l050 pla               Zeichen aus Befehlswort-Tab.
04701  517d  10 f4             bpl p1l040        Text Befehlswort-Tab. überlesen
04702  517f  30 ef             bmi p1l030        immer
04703  5181
04704  5181
```

```
04705  5181  ===> Ausgabe Text zu Token <===
04706  5181
04707  5181  c8          p1l060 iny
04708  5182  b1 24       p1l070 lda (index1),y   Zeichen aus Befehlswort-Tab.
04709  5184  30 a8              bmi p1l010       Ausg. letztes Zeichen, LIST weiter
04710  5186
04711  5186  20 0c 56           jsr outdo        Ausgabe ac auf akt. Kanal
04712  5189  d0 f6              bne p1l060       immer
04713  518b
04714  518b
04715  518b  60          p1lrts rts              Ende LIST akt. Zeile
04716  518c
04717  518c
04718  518c  ===> Wandlung ESC COMMAND Token/Text <===
04719  518c
04720  518c  aa          prescc tax              ESC COMMAND Token
04721  518d
04722  518d  c8                 iny              Zeiger auf zweiten Token
04723  518e  20 ec 42          jsr indlo0       zweiten Token nach ac
04724  5191  f0 9f             beq p1l015       Zeilenende, LIST weiter
04725  5193
04726  5193  84 4b             sty lstpnt       Zeiger in akt. Zeile
04727  5195
04728  5195  c9 02             cmp #_lecmt      niedrigster ESC COMMAND Token
04729  5197  90 27             bcc prforn       ind. Wandlg. user-def. Token/Text
04730  5199
04731  5199  c9 27             cmp #_hecmt+1    höchster+1 ESC COMMAND Token
04732  519b  b0 23             bcs prforn       ind. Wandlg. user-def. Token/Text
04733  519d
04734  519d  69 7e             adc #_tkofs-_lecmt
04735  519f  aa                 tax
04736  51a0
04737  51a0  a0 09             ldy #<esccmd     BASIC ESC COMMAND Tabelle
04738  51a2  a9 46             lda #>esccmd     BASIC ESC COMMAND Tabelle
04739  51a4  d0 c4             bne p1l026       Ausg. Text Befehlswort-Tab. ac/yr
04740  51a6
04741  51a6
04742  51a6  ===> Wandlung ESC FUNCTION Token/Text <===
04743  51a6
04744  51a6  aa          prescf tax              ESC FUNCTION Token
04745  51a7
04746  51a7  c8                 iny              Zeiger auf zweiten Token
04747  51a8  20 ec 42          jsr indlo0       zweiten Token nach ac
04748  51ab  f0 85             beq p1l015       Zeilenende, LIST weiter
04749  51ad
04750  51ad  84 4b             sty lstpnt       Zeiger in akt. Zeile
04751  51af
04752  51af  c9 02             cmp #_lefnt      niedrigster ESC FUNCTION Token
04753  51b1  90 0d             bcc prforn       ind. Wandlg. user-def. Token/Text
04754  51b3
04755  51b3  c9 0b             cmp #_hefnt+1    höchster ESC FUNCTION Token
04756  51b5  b0 09             bcs prforn       ind. Wandlg. user-def. Token/Text
04757  51b7
04758  51b7  69 7e             adc #_tkofs-_lecmt
04759  51b9  aa                 tax
04760  51ba
04761  51ba  a0 c9             ldy #<escfn      BASIC ESC FUNCTION Tabelle
04762  51bc  a9 46             lda #>escfn      BASIC ESC FUNCTION Tabelle
```

```
04763  51be  d0 aa            bne p1l026       Ausg. Text Befehlswort-Tab. ac/yr
04764  51c0
04765  51c0
04766  51c0
04766  51c0  ===> ind. Wandlg. user-def. Token/Text <===
04767  51c0
04768  51c0  e0 fe      prforn cpx #_ecmdt      ESC COMMAND Token
04769  51c2  d0 03             bne nortok       ESC FUNCTION Token
04770  51c4
04771  51c4  a2 00             ldx #0           Flag für ESC COMMAND Token
04772  51c6  2c                .byte _skip2
04773  51c7
04774  51c7
04775  51c7  ===> Token-Umwandlung <===
04776  51c7
04777  51c7  a2 ff      nortok ldx #$ff         Flag für ESC FUNCTION Token
04778  51c9  38                sec              Flag für nicht gefunden
04779  51ca  6c 0e 03          jmp (iescpr)     ind. Wandlg. user-def. Token/Text
                                                ($51cd)
04780  51cd
04781  51cd  ===> Wandlung user-def. Token/Text <===
04782  51cd
04783  51cd   Die Routine zur Wandlung user-def. Tokens in Text kann sich
04784  51cd   der Routine p1l026 ($516a) bedienen, um aus einer user-def.
04785  51cd   Befehlswort-Tab. (nach Muster) Texte zu wandeln.
04786  51cd
04787  51cd   Einsprung: xr: Nummer+$7f Text in Tab.
04788  51cd              yr: ADDRL user-def. Befehlswort-Tab.
04789  51cd              ac: ADDRH user-def. Befehlswort-Tab.
04790  51cd
04791  51cd   Ausgang: automatisch nach p1l010 ($512e)
04792  51cd
04793  51cd   Eine andere Variante übergibt in index1 ($24/$25) die Adresse
04794  51cd   des Textes zum user-def. Token und gibt mit CARRY = 0 über
04795  51cd   nescpr ($51cd) den Text aus, CARRY=1, nicht gefunden:
04796  51cd
04797  51cd
04798  51cd  ===> Wandlung user-def. Token/Text <===
04799  51cd
04800  51cd  b0 04      nescpr bcs escp10        nicht gefunden
04801  51cf
04802  51cf  a0 00             ldy #0
04803  51d1  f0 af             beq p1l070       Ausgabe Text zu Token
04804  51d3
04805  51d3
04806  51d3  4c 32 51   escp10 jmp p1l015       Ausgabe ac, LIST weiter
04807  51d6
04808  51d6
04809  51d6  ===> BASIC-Statement NEW ($a2) <===
04810  51d6
04811  51d6  f0 01      new    beq scrtch       Trennzeichen hinter NEW
04812  51d8  60               rts
04813  51d9
04814  51d9
04815  51d9  ===> BASIC-Zeiger rücksetzen NEW <===
04816  51d9
04817  51d9  a9 00      scrtch lda #0
04818  51db
```

```
04819  51db  ===> Linkadresse PGM-Ende Anfang PGM-RAM <===
04820  51db
04821  51db  a8              tay             Zeiger ADDRL Linkadresse
04822  51dc  91 2d           sta (txttab),y  ADDRL Linkadresse =0
04823  51de
04824  51de  c8              iny             Zeiger ADDRH Linkadresse
04825  51df  91 2d           sta (txttab),y  ADDRH Linkadresse =0
04826  51e1
04827  51e1  8d 6f 11        sta trcflg      TRACE aus
04828  51e4
04829  51e4  ===> Obergrenze PGM nachführen <===
04830  51e4
04831  51e4  a5 2d           lda txttab      ADDRL Untergrenze akt. PGM
04832  51e6  18              clc
04833  51e7  69 02           adc #2
04834  51e9  8d 10 12        sta txttop      ADDRL Obergrenze akt. PGM
04835  51ec
04836  51ec  a5 2e           lda txttab+1    ADDRH Untergrenze akt. PGM
04837  51ee  69 00           adc #0
04838  51f0  8d 11 12        sta txttop+1    ADDRH Obergrenze akt. PGM
04839  51f3
04840  51f3
04841  51f3  ===> BASIC-Zeiger initialisieren, CLR <===
04842  51f3
04843  51f3  20 54 52  runc  jsr stxtpt      txtptr auf PGM Anf.
04844  51f6
04845  51f6  a9 00           lda #0          ZERO-Flag setzen
04846  51f8
04847  51f8
04848  51f8  ===> BASIC-Statement CLR ($9c) <===
04849  51f8
04850  51f8  d0 55     clear bne stkrts      kein Trennzeichen, Abbruch
04851  51fa
04852  51fa
04853  51fa  ===> CLR durchführen <===
04854  51fa
04855  51fa  20 7b 92  clearc jsr cclall     off. Files :=0, Ersatz-I/O
04856  51fd
04857  51fd  a0 00           ldy #0
04858  51ff  84 7a           sty dsdesc      Desc. DS$ (Länge)
04859  5201
04860  5201  88              dey             $ff
04861  5202  8c 0c 12        sty trapno+1    high Zeilennr. TRAP Routine
04862  5205  8c 09 12        sty errlin      Low-Byte Zeilennummer für RESUME
04863  5208  8c 0a 12        sty errlin+1    High-Byte Zeilennummer für RESUME
04864  520b  8c 08 12        sty errnum      >127: Fehlernummer für »ready.«
04865  520e
04866  520e  a5 39           lda mxmem1      ADDRL Obergrenze VAR RAM
04867  5210  a4 3a           ldy mxmem1+1    ADDRH Obergrenze VAR RAM
04868  5212
04869  5212  85 35           sta fretop      ADDRL Zeiger Anfang String-Variable
04870  5214  84 36           sty fretop+1    ADDRH Zeiger Anfang String-Variable
04871  5216
04872  5216  a9 ff           lda #<stkbot    Untergrenze BASIC-Stapel
04873  5218  a0 09           ldy #>stkbot    Untergrenze BASIC-Stapel
04874  521a
04875  521a  85 7d           sta tos         ADDRL BASIC-Stapel Zeiger
04876  521c  84 7e           sty tos+1       ADDRH BASIC-Stapel Zeiger
```

```
04877  521e
04878  521e  a5 2f           lda vartab       ADDRL Zeiger Anfang einf. Variable
04879  5220  a4 30           ldy vartab+1     ADDRH Zeiger Anfang einf. Variable
04880  5222
04881  5222  85 31           sta arytab       ADDRL Zeiger Anfang Feld-Tabelle
04882  5224  84 32           sty arytab+1     ADDRH Zeiger Anfang Feld-Tabelle
04883  5226
04884  5226  85 33           sta strend       ADDRL+1 Obergr. ARRAY Tabelle
04885  5228  84 34           sty strend+1     ADDRH+1 Obergr. ARRAY Tabelle
04886  522a
04887  522a  ===> Vorgabewerte PRINT USING eintragen <===
04888  522a
04889  522a  a2 03           ldx #3
04890  522c  bd 50 52  clrpu  lda pudefs,x     PRINT USING Zeichen
04891  522f  9d 04 12         sta pufill,x     PRINT USING: Füllzeichen
04892  5232  ca               dex
04893  5233  10 f7           bpl clrpu
04894  5235
04895  5235
04896  5235  ===> reset read-, Stapel- und Befehlszeiger <===
04897  5235
04898  5235  20 e1 5a  fload  jsr resto1       READ-Zeiger auf BASIC-Anfang
04899  5238
04900  5238
04901  5238  ===> reset Stapel- u. BASIC-Befehl-Zeiger <===
04902  5238
04903  5238  a2 1b     stkini ldx #<tempst     Descriptor-Stapel
04904  523a  86 18            stx temppt       Zeiger in den String-Stapel
04905  523c
04906  523c  68               pla
04907  523d  a8               tay              Rücksprung ADDRL
04908  523e
04909  523e  68               pla              Rücksprung ADDRH
04910  523f
04911  523f  a2 fa            ldx #256-_hedln-2
04912  5241  9a               txs
04913  5242
04914  5242  48               pha              Rücksprung ADDRH
04915  5243
04916  5243  98               tya
04917  5244  48               pha              Rücksprung ADDRL
04918  5245
04919  5245  a9 00            lda #0
04920  5247  8d 03 12         sta oldtxt+1     CONT verhindern
04921  524a  85 12            sta sublfg       INT Var. zulassen
04922  524c  8d df 03         sta bits         FAC Overflow Bit
04923  524f
04924  524f  60         stkrts rts
04925  5250
04926  5250
04927  5250  ===> PRINT USING Zeichen <===
04928  5250
04929  5250  20 2c 2e  pudefs .byte ' ,.$'
04929  5253  24
04930  5254
04931  5254
```

```
04932  5254  ===> txtptr auf PGM Anf. <===
04933  5254
04934  5254  18          stxtpt clc
04935  5255  a5 2d              lda txttab       ADDRL Untergrenze akt. PGM
04936  5257  69 ff             adc #$ff         -1
04937  5259  85 3d             sta txtptr       ADDRL chrget-Zeiger
04938  525b
04939  525b  a5 2e             lda txttab+1     ADDRH Untergrenze akt. PGM
04940  525d  69 ff             adc #$ff
04941  525f  85 3e             sta txtptr+1     ADDRH chrget-Zeiger
04942  5261  60                rts
04943  5262
04944  5262
04945  5262  ===> BASIC-Statement RETURN ($8e) <===
04946  5262
04947  5262  68          return pla
04948  5263  68                 pla
04949  5264
04950  5264  a9 8d              lda #_gosut      GOSUB Token
04951  5266  20 aa 4f           jsr search       Token (ac) in BASIC-Stapel suchen
04952  5269  f0 05              beq ret010       GOSUB Token gefunden
04953  526b
04954  526b  ===> Behandlung »?return without gosub« <===
04955  526b
04956  526b  a2 0c              ldx #_errg
04957  526d  4c 3c 4d           jmp error        ind. Behandlung Fehlermeldung
04958  5270
04959  5270
04960  5270  ===> txtptr hinter aufrufendes GOSUB pos. <===
04961  5270
04962  5270  20 50 50    ret010 jsr movfnd       Kopie Suchzeiger nach Stapelzeiger
04963  5273
04964  5273  a0 05              ldy #_lngos      Stapel-Bytes für GOSUB
04965  5275  20 59 50           jsr rlsstk       BASIC-Stapel Zeiger um yr erhöhen
04966  5278
04967  5278  88                 dey
04968  5279  b1 3f              lda (fndpnt),y   ADDRH chrget vor gosub
04969  527b  85 3e             sta txtptr+1     ADDRH chrget-Zeiger
04970  527d
04971  527d  88                 dey
04972  527e  b1 3f              lda (fndpnt),y   ADDRL chrget vor GOSUB
04973  5280  85 3d             sta txtptr       ADDRL chrget-Zeiger
04974  5282
04975  5282  88                 dey
04976  5283  b1 3f              lda (fndpnt),y   High-Byte Zeilennr. vor GOSUB
04977  5285  20 3b a8           jsr retpat       High-Byte Zeilennr./Direktmod. eintr.
04978  5288
04979  5288  b1 3f              lda (fndpnt),y   Low-Byte Zeilennr. vor GOSUB
04980  528a  85 3b             sta curlin       Low-Byte akt. Zeilennummer
04981  528c
04982  528c  4c 8f 52           jmp data         txtptr auf Trennzeichen hinter GOSUB
04983  528f
04984  528f
04985  528f  ===> BASIC-Statement DATA ($83) <===
04986  528f
04987  528f  20 a2 52    data   jsr datan        yr hinter nächst. »:« bzw. $00
04988  5292
04989  5292
```

```
04990  5292  ===> yr zu txtptr addieren <===
04991  5292
04992  5292  98          addon  tya              Zeiger hinter »:« bzw. $00
04993  5293  18                 clc
04994  5294  65 3d             adc txtptr       ADDRL chrget-Zeiger
04995  5296  85 3d             sta txtptr       ADDRL chrget-Zeiger
04996  5298
04997  5298  90 02             bcc remrts
04998  529a  e6 3e             inc txtptr+1     ADDRH chrget-Zeiger
04999  529c  60          remrts rts
05000  529d
05001  529d
05002  529d  ===> BASIC-Statement REM ($8f) <===
05003  529d
05004  529d  20 a5 52    rem    jsr remn         BASIC-Zeilenende suchen, Offset yr
05005  52a0  f0 f0              beq addon        yr zu txtptr addieren
05006  52a2
05007  52a2
05008  52a2  ===> yr hinter nächst. »:« bzw. $00 <===
05009  52a2
05010  52a2  a2 3a       datan  ldx #':'         Trennzeichen
05011  52a4  2c                 .byte _skip2
05012  52a5
05013  52a5
05014  52a5  ===> BASIC-Zeilenende suchen, Offset yr <===
05015  52a5
05016  52a5  a2 00       remn   ldx #$00         Endezeichen PGM-Zeile
05017  52a7  86 09              stx charac       Suchzeichen
05018  52a9
05019  52a9  a0 00              ldy #$00         Endezeichen PGM-Zeile
05020  52ab  84 0a              sty endchr       Abbruchzeichen
05021  52ad
05022  52ad  a5 0a       exchqt lda endchr       Abbruchzeichen
05023  52af  a6 09              ldx charac       und Suchzeichen
05024  52b1
05025  52b1  85 09              sta charac       vertauschen
05026  52b3  86 0a              stx endchr
05027  52b5
05028  52b5  20 c9 03    remer  jsr indtxt       Zeichen aus PGM-Zeile
05029  52b8  f0 e2              beq remrts       Abbruchzeichen
05030  52ba
05031  52ba  c5 0a              cmp endchr       Suchzeichen
05032  52bc  f0 de              beq remrts
05033  52be
05034  52be  c8                 iny              Zeiger in PGM-Zeile
05035  52bf  c9 22              cmp #'"'
05036  52c1  d0 f2              bne remer
05037  52c3  f0 e8              beq exchqt       Suchz. in Anführg.zeich. ignorieren
05038  52c5
05039  52c5
05040  52c5  ===> BASIC-Statement IF ($8b) <===
05041  52c5
05042  52c5  20 ef 77    if     jsr frmevl       Wahrheitswert IF-Bedingung nach
                                                 facexp
05043  52c8
05044  52c8  20 86 03           jsr chrgot       Zeichen hinter IF-Bedingung
05045  52cb  c9 89              cmp #_gotot      GOTO Token
05046  52cd  f0 05              beq okgoto       Test Wahrheitswert IF-Bedingung
```

```
05047  52cf
05048  52cf  a9 a7            lda #_thent      THEN Token
05049  52d1  20 5e 79         jsr synchr       Test auf Zeichen in ac, sonst Fehler
05050  52d4
05051  52d4
05052  52d4  ===> Test Wahrheitswert IF-Bedingung <===
05053  52d4
05054  52d4  a5 63      okgoto lda facexp      Wahrheitswert IF-Bedingung
05055  52d6  d0 26            bne iftrue       wahr
05056  52d8
05057  52d8
05057  52d8  ===> IF-Bedingung nicht erfüllt <===
05058  52d8
05059  52d8  20 86 03         jsr chrgot       letztes Zeichen erneut nach ac
05060  52db  c9 fe            cmp #_ecmdt      ESC COMMAND Token
05061  52dd  d0 0b            bne lkelse       Test auf ELSE nach IF
05062  52df
05063  52df  c8               iny
05064  52e0  20 c9 03         jsr indtxt       Zeichen hinter ESC COMMAND Token
05065  52e3  c9 18            cmp #_begnt      BEGIN Token
05066  52e5  d0 03            bne lkelse       Test auf ELSE nach IF
05067  52e7
05068  52e7  20 20 53         jsr fndbnd       BEGIN bis BEND überlesen
05069  52ea
05070  52ea
05071  52ea  ===> Test auf ELSE nach IF <===
05072  52ea
05073  52ea  20 8f 52   lkelse jsr data        Zeile bis Trennzeichen lesen
05074  52ed
05075  52ed  a0 00            ldy #0
05076  52ef  20 c9 03         jsr indtxt       Trennzeichen
05077  52f2  f0 a9            beq rem          Zeilenende, kein ELSE
05078  52f4
05079  52f4  20 80 03         jsr chrget       nächstes Zeichen nach ac
05080  52f7  c9 d5            cmp #_elset      ELSE Token
05081  52f9  d0 ef            bne lkelse       Test auf ELSE nach IF
05082  52fb
05083  52fb  ===> ELSE nach nicht erfüllter IF-Bedingung <===
05084  52fb
05085  52fb  20 80 03         jsr chrget       nächstes Zeichen nach ac
05086  52fe
05087  52fe
05088  52fe  ===> Bearbeit. Anweis. nach IF/ELSE <===
05089  52fe
05090  52fe  20 86 03   iftrue jsr chrgot      letztes Zeichen erneut nach ac
05091  5301  f0 17            beq iftr20       Zeilenende
05092  5303  b0 03            bcs iftr10       keine Ziffer
05093  5305
05094  5305  ===> Zeilennummer (GOTO) nach IF/ELSE <===
05095  5305
05096  5305  4c db 59         jmp goto         BASIC-Statement GOTO ($89)
05097  5308
05098  5308
05099  5308  c9 fe      iftr10 cmp #_ecmdt      ESC COMMAND Token
05100  530a  d0 0e            bne iftr20       kein BEGIN-BEND
05101  530c
05102  530c  c8               iny
05103  530d  20 c9 03         jsr indtxt       Zeichen hinter ESC COMMAND Token
```

```
05104  5310
05105  5310  c9 18           cmp #_begnt      BEGIN Token
05106  5312  d0 06           bne iftr20       kein BEGIN-BEND
05107  5314
05108  5314  20 80 03        jsr chrget       BEGIN Token
05109  5317  20 80 03        jsr chrget       Zeichen hinter BEGIN Token
05110  531a
05111  531a  20 86 03  iftr20 jsr chrgot       letztes Zeichen erneut nach ac
05112  531d  4c 3f 4b        jmp xeqcm1       BASIC-Befehl (Token in ac) ausführen
05113  5320
05114  5320
05115  5320
05115  5320  ===> BEGIN bis BEND überlesen <===
05116  5320
05117  5320  20 80 03  fndbnd jsr chrget       nächstes Zeichen nach ac
05118  5323  d0 27           bne fndb30
05119  5325
05120  5325  c9 3a      fndb10 cmp #':'        Trennzeichen
05121  5327  f0 f7           beq fndbnd       Zeilenende suchen
05122  5329
05123  5329
05124  5329  ===> Parameter nächste PGM-Zeile <===
05125  5329
05126  5329  24 7f      fndb20 bit runmod      Test Direktmodus
05127  532b  10 4a           bpl bnderr       Behandlung »?bend not found error«
05128  532d
05129  532d  ===> Test Programmende <===
05130  532d
05131  532d  a0 02           ldy #2
05132  532f  20 c9 03        jsr indtxt       ADDRH Linkadresse
05133  5332  f0 43           beq bnderr       Behandlung »?bend not found error«
05134  5334
05135  5334  ===> nächste Zeilennummer nach curlin <===
05136  5334
05137  5334  c8               iny
05138  5335  20 c9 03        jsr indtxt       Low-Byte nächste Zeilennr.
05139  5338  85 3b           sta curlin       Low-Byte akt. Zeilennummer
05140  533a
05141  533a  c8               iny
05142  533b  20 c9 03        jsr indtxt       High-Byte nächste Zeilennr.
05143  533e  85 3c           sta curlin+1     High-Byte akt. Zeilennummer
05144  5340
05145  5340  ===> txtptr in nächste PGM-Zeile <===
05146  5340
05147  5340  98               tya
05148  5341  18               clc
05149  5342  65 3d           adc txtptr       ADDRL chrget-Zeiger
05150  5344  85 3d           sta txtptr       ADDRL chrget-Zeiger
05151  5346
05152  5346  90 d8           bcc fndbnd       BEGIN bis BEND überlesen
05153  5348  e6 3e           inc txtptr+1     ADDRH chrget-Zeiger
05154  534a  d0 d4           bne fndbnd       immer
05155  534c
05156  534c
05157  534c  c9 22      fndb30 cmp #«"«
05158  534e  d0 07           bne fndb40       Test auf REM Token
05159  5350
```

```
05160  5350  ===> Text in "" überlesen <===
05161  5350
05162  5350  20 7c 53          jsr unquot      Text in "" überlesen
05163  5353  f0 d0             beq fndb10      Trennzeichen
05164  5355  d0 c9             bne fndbnd      Zeilenende suchen
05165  5357
05166  5357
05167  5357  ===> Test auf REM Token <===
05168  5357
05169  5357  c9 8f       fndb40 cmp #_remt      REM Token
05170  5359  d0 06             bne fndb50      Test auf BEND Token
05171  535b
05172  535b  20 9d 52          jsr rem         bis Zeilenende überlesen
05173  535e  4c 29 53          jmp fndb20      Parameter nächste PGM-Zeile
05174  5361
05175  5361
05176  5361  ===> Test auf BEND Token <===
05177  5361
05178  5361  c9 fe       fndb50 cmp #_ecmdt     ESC COMMAND Token
05179  5363  d0 bb             bne fndbnd      Zeilenende suchen
05180  5365
05181  5365  20 80 03          jsr chrget      Zeichen hinter ESC COMMAND Token
05182  5368  c9 19             cmp #_bendt     BEND Token
05183  536a  f0 0a             beq fndbrt      BEND gefunden, RTS
05184  536c
05185  536c  ===> Test auf verschachtelte BEGIN-BEND <===
05186  536c
05187  536c  c9 18             cmp #_begnt     BEGIN Token
05188  536e  d0 b0             bne fndbnd      BEGIN bis BEND überlesen
05189  5370
05190  5370  20 20 53          jsr fndbnd      rekursiver Aufruf: BEND suchen
05191  5373  4c 20 53          jmp fndbnd      nächsthöheres BEND weitersuchen
05192  5376
05193  5376
05194  5376  60          fndbrt rts
05195  5377
05196  5377
05197  5377  ===> Behandlung »?bend not found error' <===
05198  5377
05199  5377  a2 25       bnderr ldx #_ernob
05200  5379  4c 3c 4d          jmp error       ind. Behandlung Fehlermeldung
05201  537c
05202  537c
05203  537c  ===> Text in "" überlesen <===
05204  537c
05205  537c  a0 00       unquot ldy #0          Zeiger in PGM-Zeile
05206  537e
05207  537e
05208  537e  ===> chrget-Zeiger erhöhen <===
05209  537e
05210  537e  e6 3d       unqu10 inc txtptr      ADDRL chrget-Zeiger
05211  5380  d0 02             bne unqu20
05212  5382  e6 3e             inc txtptr+1    ADDRH chrget-Zeiger
05213  5384  20 c9 03   unqu20 jsr indtxt      Zeichen aus PGM-Zeile
05214  5387  f0 07             beq unqurt      Zeilenende
05215  5389
05216  5389  c9 22             cmp #«"«
05217  538b  d0 f1             bne unqu10      chrget-Zeiger erhöhen
```

```
05218  538d
05219  538d  4c 80 03          jmp chrget       Zeichen hinter "" nach ac
05220  5390
05221  5390
05222  5390  60         unqurt rts
05223  5391
05224  5391
05225  5391  ===> BASIC-Statement ELSE ($d5) <===
05226  5391
05227  5391  c9 fe      else   cmp #_ecmdt      ESC COMMAND Token
05228  5393  d0 0b             bne elsrem       folgt nicht BEGIN
05229  5395
05230  5395
05230  5395  ===> Test auf BEGIN-BEND <===
05231  5395
05232  5395  c8                iny
05233  5396  20 c9 03          jsr indtxt       Zeichen hinter ESC COMMAND Token
05234  5399
05235  5399  c9 18             cmp #_begnt      BEGIN Token
05236  539b  d0 03             bne elsrem       folgt nicht BEGIN
05237  539d
05238  539d  20 20 53          jsr fndbnd       BEGIN bis BEND überlesen
05239  53a0  4c 9d 52   elsrem jmp rem          Zeilenrest überlesen
05240  53a3
05241  53a3   Das Statement ELSE wird nur aufgerufen, wenn die IF-Bedingung
05242  53a3   wahr ist - daher wird hier alles überlesen.
05243  53a3   Statements hinter ELSE werden abgearbeitet, wenn die IF-Bedingung
05244  53a3   falsch ist: lkelse $52ea.
05245  53a3
05246  53a3
05247  53a3  ===> BASIC-Statement ON ($91) <===
05248  53a3
05249  53a3  20 f4 87   ongoto jsr getbyt       Parameter hinter ON nach faclo
05250  53a6
05251  53a6  ===> Test auf folg. GOSUB/GOTO <===
05252  53a6
05253  53a6  48                pha              Zeichen hinter Parameter
05254  53a7
05255  53a7  c9 8d             cmp #_gosut      GOSUB Token
05256  53a9  f0 07             beq onglop
05257  53ab
05258  53ab  c9 89             cmp #_gotot      GOTO Token
05259  53ad  f0 03             beq onglop
05260  53af
05261  53af  ===> Behandlung »?syntax error« <===
05262  53af
05263  53af  4c 6c 79          jmp snerr        Behandlung »?syntax error«
05264  53b2
05265  53b2
05266  53b2  c6 67      onglop dec faclo        ON Parameter
05267  53b4  d0 04             bne onglp1       nächste Zeilennr. lesen
05268  53b6
05269  53b6  68                pla              GOSUB/GOTO Token
05270  53b7  4c 59 4b          jmp xeqcm2       BASIC-Token prüfen
05271  53ba
05272  53ba
```

```
05273  53ba  ===> nächste Zeilennr. lesen <===
05274  53ba
05275  53ba  20 80 03   onglp1 jsr chrget       nächstes Zeichen nach ac
05276  53bd  20 a0 50          jsr linget       Zeilennr. nach linnum
05277  53c0
05278  53c0  c9 2c             cmp #','
05279  53c2  f0 ee             beq onglop
05280  53c4
05281  53c4  68                pla              GOSUB/GOTO Token
05282  53c5  60                rts
05283  53c6
05284  53c6
05285  53c6                    .end
05286  53c6                    .lib basic.varia
05287  53c6  put "@basic.varia"
```

```
05289  53c6
05290  53c6
05291  53c6  ===> BASIC-Statement LET ($88) <===
05292  53c6
05293  53c6  20 af 7a   let    jsr ptrget       folg. Variable suchen/anlegen
05294  53c9  85 4b             sta lstpnt       ADDRL Var.
05295  53cb  84 4c             sty lstpnt+1     ADDRH Var.
05296  53cd
05297  53cd  a9 b2             lda #_eqult      GLEICH = Token
05298  53cf  20 5e 79          jsr synchr       Test auf Zeichen in ac, sonst Fehler
05299  53d2
05300  53d2  a5 10             lda intflg       Flag für Gleitkomma/Integer
05301  53d4  48                pha
05302  53d505303  53d5  a5 0f              lda valtyp       Flag Variablentyp
05304  53d7  48                pha
05305  53d8
05306  53d8  20 ef 77          jsr frmevl       folgenden Ausdruck auswerten
05307  53db
05308  53db  68                pla              Variablen Typ
05309  53dc  2a                rol a            Bit7 ins Carry (1=String)
05310  53dd
05311  53dd  20 de 77          jsr chkval       Test VAR Typ (CARRY=1:String,
                                                =0:num)
05312  53e0  d0 22             bne copstr       Stringzuweisung durch LET
05313  53e2
05314  53e2  68                pla              Flag für Gleitkomma/Integer
05315  53e3
05316  53e3
05317  53e3  ===> GK/INT aus FAC nach Var. <===
05318  53e3
05319  53e3  10 15       qintgr bpl copflt       GK aus FAC nach Var.
05320  53e5
05321  53e5  ===> FAC nach INT, INT nach Var. <===
05322  53e5
05323  53e5  20 47 8c          jsr round        FAC runden
05324  53e8  20 b4 84          jsr ayint        Wandlung FAC nach sgnd. INT
05325  53eb
05326  53eb  a0 00             ldy #0
05327  53ed  a5 66             lda facmo        FAC: M2
05328  53ef
05329  53ef  8d 04 ff          sta lcrd         ROM / VAR-RAM einschalten
05330  53f2  91 4b             sta (lstpnt),y
05331  53f4
05332  53f4  c8                iny
05333  53f5  a5 67             lda faclo        FAC: M1
05334  53f7  91 4b             sta (lstpnt),y
05335  53f9
05336  53f9  60                rts
05337  53fa
05338  53fa
05339  53fa  ===> GK aus FAC nach Var. <===
05340  53fa
05341  53fa  a6 4b       copflt ldx lstpnt       ADDRL Var.
05342  53fc  a4 4c              ldy lstpnt+1     ADDRH Var.
05343  53fe  8d 04 ff          sta lcrd         ROM / VAR-RAM einschalten
05344  5401  4c 00 8c          jmp movmf        FAC runden, nach RAM xr/yr
05345  5404
05346  5404
```

```
05347  5404  ===> Stringzuweisung durch LET <===
05348  5404
05349  5404  68         copstr pla              intflg verwerfen
05350  5405
05351  5405   Kommentare beziehen sich auf den allgemeinen Fall n$ = o$
05352  5405
05353  5405
05354  5405  ===> String zuweisen <===
05355  5405
05356  5405  a4 4c      inpcom ldy lstpnt+1     ADDRH n$
05357  5407  c0 03             cpy #>zero       Konstante Null
05358  5409  d0 72             bne getspt       zeigt nicht auf Konstante Null
05359  540b
05360  540b  ===> Test / Zuweisung TI$ = o$ <===
05361  540b
05362  540b  20 81 87          jsr frefac       Adresse o$ nach index1, Länge ac
05363  540e  c9 06             cmp #_tidln      Länge TI$-String
05364  5410  d0 3e             bne fcerr2       Behandlung »?illegal quantity error«
05365  5412
05366  5412  a0 00             ldy #0
05367  5414  84 63             sty facexp       FAC: EX
05368  5416  84 68             sty facsgn       FAC: SG
05369  5418
05370  5418
05371  5418  ===> TI$ als Systemzeit abspeichern <===
05372  5418
05373  5418  84 72      timelp sty fbufpt       Zeiger in TI$-String
05374  541a
05375  541a  ===> höherwertiges Byte aus TI$-String <===
05376  541a
05377  541a  20 48 54          jsr timnum       FAC plus INT ASCII aus TI$-String
05378  541d  20 17 8b          jsr mul10        FAC mit 10 multiplizieren
05379  5420  e6 72             inc fbufpt       Zeiger in TI$-String
05380  5422
05381  5422  ===> niederwertiges Byte aus TI$-String <===
05382  5422
05383  5422  a4 72             ldy fbufpt       Zeiger in TI$-String
05384  5424  20 48 54          jsr timnum       FAC plus INT ASCII aus TI$-String
05385  5427  20 38 8c          jsr movaf        FAC runden, FAC nach ARG
05386  542a  aa                tax              ARG EX
05387  542b  f0 05             beq noml6        nicht mit 6 mult.
05388  542d
05389  542d  ===> FAC mal sechs <===
05390  542d
05391  542d  e8                inx              ARG EX mal zwei
05392  542e  8a                txa
05393  542f  20 22 8b          jsr finml6       (FAC plus ARG) mal zwei
05394  5432
05395  5432  a4 72      noml6  ldy fbufpt       ADDRL Zeiger in Puffer
05396  5434  c8                iny
05397  5435  c0 06             cpy #_tidln      Länge TI$-String
05398  5437  d0 df             bne timelp       TI$ als Systemzeit abspeichern
05399  5439
05400  5439  20 17 8b          jsr mul10        FAC mit 10 multiplizieren
05401  543c  20 c7 8c          jsr qint         FAC GK nach INT
05402  543f
05403  543f  a6 66             ldx facmo        FAC: M2
05404  5441  a4 65             ldy facmoh       FAC: M3
```

```
05405  5443  a5 67            lda faclo        FAC: M1
05406  5445  4c db ff         jmp ksttim       System-Uhr setzen (Sprungv.)
05407  5448
05408  5448
05409  5448  ===> FAC plus INT ASCII aus TI$-String <===
05410  5448
05411  5448  20 b7 03   timnum jsr ind1_1      LDA (index1),y aus VAR-RAM
05412  544b  20 90 03         jsr qnum         Test ac numerisch (CARRY=0)
05413  544e  90 03            bcc gotnum
05414  5450
05415  5450  ===> Behandlung »?illegal quantity error« <===
05416  5450
05417  5450  4c 28 7d   fcerr2 jmp fcerr       Behandlung »?illegal quantity error«
05418  5453
05419  5453
05420  5453  e9 2f      gotnum sbc #$30-1      (CARRY=0)
05421  5455  4c b0 8d         jmp finlog       FAC plus INT in ac
05422  5458
05423  5458
05424  5458  68         dskx1  pla              ADDRH vom Stapel
05425  5459  c8                iny              Zeiger wieder auf ADDRH
05426  545a
05427  545a
05428  545a  ===> Test auf Lage des Quellstrings o$  <===
05429  545a
05430  545a  c5 36      dskx2  cmp fretop+1     ADDRH Zeiger Anfang String-Variable
05431  545c  90 18            bcc dntcpy       unterhalb Stringbereich
05432  545e  d0 08            bne qvaria
05433  5460
05434  5460  88                dey
05435  5461  20 e7 42         jsr indfmo       LDA (facmo),y aus VAR-RAM
05436  5464  c5 35            cmp fretop       ADDRL Zeiger Anfang String-Variable
05437  5466  90 0e            bcc dntcpy
05438  5468
05439  5468  a4 67      qvaria ldy faclo        FAC: M1
05440  546a  c4 30            cpy vartab+1     ADDRH Zeiger Anfang einf. Variable
05441  546c  90 08            bcc dntcpy
05442  546e  d0 24            bne copy         String ins RAM, Test/Trailer
05443  5470
05444  5470  a5 66            lda facmo        FAC: M2
05445  5472  c5 2f            cmp vartab       ADDRL Zeiger Anfang einf. Variable
05446  5474  b0 1e            bcs copy         String ins RAM, Test/Trailer
05447  5476
05448  5476  a5 66      dntcpy lda facmo        FAC: M2
05449  5478  a4 67            ldy faclo        FAC: M1
05450  547a  4c b2 54         jmp copyc
05451  547d
05452  547d
05453  547d  ===> Test auf (gült.) DS$-String <===
05454  547d
05455  547d  a0 02      getspt ldy #2           Zeiger auf ADDRH String
05456  547f  20 e7 42         jsr indfmo       ADDRH nach ac
05457  5482  c5 7c            cmp dsdesc+2     ADDRH DS$-String
05458  5484  d0 d4            bne dskx2        nicht DS$
05459  5486
05460  5486  48                pha              ADDRH String
05461  5487
05462  5487  88                dey              Zeiger auf ADDRL String
```

```
05463  5488  20 e7 42          jsr indfmo       ADDRL nach ac
05464  548b  c5 7b             cmp dsdesc+1     ADDRL DS$-String
05465  548d  d0 c9             bne dskx1        nicht DS$
05466  548f
05467  548f  a5 7a             lda dsdesc       Länge DS$-String
05468  5491  f0 c5             beq dskx1        =0
05469  5493
05470  5493  ===> DS$ zuweisen <===
05471  5493
05472  5493  68                pla
05473  5494
05474  5494
05475  5494  ===> String ins RAM, Test/Trailer <===
05476  5494
05477  5494  a0 00       copy  ldy #0           Zeiger auf Stringlänge
05478  5496  20 e7 42          jsr indfmo       LDA (facmo),y aus VAR-RAM
05479  5499  20 88 86          jsr strini       Arbeitsstring Länge ac anlegen
05480  549c
05481  549c  a5 52             lda dscpnt       ADDRL Zeiger String-Descr.
05482  549e  a4 53             ldy dscpnt+1     ADDRH Zeiger String-Descr.
05483  54a0
05484  54a0  85 70             sta strng1
05485  54a2  84 71             sty strng1+1
05486  54a4
05487  54a4  20 4e 87          jsr movins       String von Descr. strng1 in
                                                Arb.string kop.
05488  54a7
05489  54a7  a5 70             lda strng1
05490  54a9  a4 71             ldy strng1+1
05491  54ab  20 e0 87          jsr fretms       Descriptor vom Stapel entfernen
05492  54ae
05493  54ae  a9 63             lda #<dsctmp     temp. Stringdescr.
05494  54b0  a0 00             ldy #>dsctmp     temp. Stringdescr.
05495  54b2
05496  54b2  85 52       copyc sta dscpnt       ADDRL Zeiger String-Descr.
05497  54b4  84 53             sty dscpnt+1     ADDRH Zeiger String-Descr.
05498  54b6
05499  54b6  85 24             sta index1
05500  54b8  84 25             sty index1+1
05501  54ba
05502  54ba  20 e0 87          jsr fretms       Descriptor vom Stapel entfernen
05503  54bd  20 f6 54          jsr stradj       Test Trailer, index1 auf Trailer
05504  54c0  90 0e             bcc olgarb       kein Trailer
05505  54c2
05506  54c2  ===> Trailer hinter String <===
05507  54c2
05508  54c2  a0 00             ldy #0
05509  54c4  a5 4b             lda lstpnt
05510  54c6  8d 04 ff          sta lcrd         ROM / VAR-RAM einschalten
05511  54c9  91 24             sta (index1),y
05512  54cb
05513  54cb  c8                iny
05514  54cc  a5 4c             lda lstpnt+1
05515  54ce  91 24             sta (index1),y
05516  54d0
05517  54d0
```

```
05518  54d0  ===> alten String ungültig machen <===
05519  54d0
05520  54d0  a5 4b      olgarb lda lstpnt
05521  54d2  85 24             sta index1
05522  54d4
05523  54d4  a5 4c             lda lstpnt+1
05524  54d6  85 25             sta index1+1
05525  54d8
05526  54d8  20 f6 54          jsr stradj       Test Trailer, index1 auf Trailer
05527  54db  90 0c             bcc cpydsc       kein Trailer
05528  54dd
05529  54dd  ===> Trailer für ungültigen String <===
05530  54dd
05531  54dd  88                dey
05532  54de  a9 ff             lda #$ff         Flag String ungültig
05533  54e0  8d 04 ff          sta lcrd         ROM / VAR-RAM einschalten
05534  54e3  91 24             sta (index1),y   letztes Trailer-Byte
05535  54e5
05536  54e5  88                dey
05537  54e6  8a                txa              Stringlänge
05538  54e7  91 24             sta (index1),y   erstes Trailer-Byte
05539  54e9
05540  54e9
05541  54e9  ===> Stringdescriptor kopieren <===
05542  54e9
05543  54e9  a0 02      cpydsc ldy #2
05544  54eb  a9 52      cpydlp lda #<dscpnt      ADDRL Zeiger String-Descr.
05545  54ed  20 ab 03          jsr iram1        LDA (ac),y aus VAR-RAM
05546  54f0  91 4b             sta (lstpnt),y
05547  54f2  88                dey
05548  54f3  10 f6             bpl cpydlp
05549  54f5
05550  54f5  60                rts
05551  54f6
05552  54f6
05553  54f6  ===> Test Trailer, index1 auf Trailer <===
05554  54f6
05555  54f6  a0 00      stradj ldy #0           Zeiger auf Stringlänge
05556  54f8  20 b7 03          jsr ind1_1       LDA (index1),y aus VAR-RAM
05557  54fb  48                pha              Stringlänge
05558  54fc  f0 39             beq sadj8        leerer String, kein Trailer
05559  54fe
05560  54fe  c8                iny              Zeiger auf ADDRL String
05561  54ff  20 b7 03          jsr ind1_1       LDA (index1),y aus VAR-RAM
05562  5502  aa                tax              ADDRL String
05563  5503
05564  5503  c8                iny              Zeiger auf ADDRH String
05565  5504  20 b7 03          jsr ind1_1       LDA (index1),y aus VAR-RAM
05566  5507
05567  5507  c5 3a             cmp mxmem1+1     ADDRH Obergrenze VAR RAM
05568  5509  90 06             bcc sadj2
05569  550b  d0 2a             bne sadj8        String ohne Trailer
05570  550d
05571  550d  e4 39             cpx mxmem1       ADDRL Obergrenze VAR RAM
05572  550f  b0 26             bcs sadj8        String ohne Trailer
05573  5511
05574  5511  20 b7 03   sadj2  jsr ind1_1       LDA (index1),y aus VAR-RAM
05575  5514
```

```
05576  5514  c5 36           cmp fretop+1     ADDRH Zeiger Anfang String-Variable
05577  5516  90 1f           bcc sadj8        String ohne Trailer
05578  5518  d0 04           bne sadj3
05579  551a
05580  551a  e4 35           cpx fretop       ADDRL Zeiger Anfang String-Variable
05581  551c  90 19           bcc sadj8        String ohne Trailer
05582  551e
05583  551e  c5 7c     sadj3 cmp dsdesc+2     Desc. DS$ (ADDRH)
05584  5520  d0 04           bne sadj4
05585  5522
05586  5522  e4 7b           cpx dsdesc+1     Desc. DS$ (ADDRL)
05587  5524  f0 11           beq sadj8        String ohne Trailer
05588  5526
05589  5526  86 24     sadj4 stx index1
05590  5528  85 25           sta index1+1
05591  552a
05592  552a  68               pla
05593  552b  aa               tax
05594  552c
05595  552c  18               clc
05596  552d  65 24            adc index1
05597  552f  85 24            sta index1
05598  5531
05599  5531  90 02           bcc sadj6        String mit Trailer
05600  5533  e6 25           inc index1+1
05601  5535
05602  5535
05603  5535  ===> String mit Trailer <===
05604  5535
05605  5535  38         sadj6 sec
05606  5536  60               rts
05607  5537
05608  5537
05609  5537  ===> String ohne Trailer <===
05610  5537
05611  5537  68         sadj8 pla
05612  5538  18               clc
05613  5539  60               rts
05614  553a
05615  553a
05616  553a                   .end
05617  553a                   .lib basic.cmd_2
05618  553a  put"@basic.cmd_2"
```

```
05620  553a
05621  553a
05622  553a  ===> BASIC-Statement PRINT# ($98) <===
05623  553a
05624  553a  20 40 55   printn jsr cmd          BASIC-Statement CMD ($9d)
05625  553d  4c 58 56          jmp iodone       akt. Kanal clr, Tastatur aktivieren
05626  5540
05627  5540
05628  5540  ===> BASIC-Statement CMD ($9d) <===
05629  5540
05630  5540  20 f4 87   cmd    jsr getbyt       8 Bit INT üb. frmnum nach xr
05631  5543  f0 05             beq saveit       Trennzeichen
05632  5545
05633  5545  a9 2c             lda #','         Separator vor Ausdrucksliste
05634  5547  20 5e 79          jsr synchr       Test auf Zeichen in ac, sonst Fehler
05635  554a
05636  554a  08         saveit php
05637  554b  86 15             stx channl       aktiver I/O-Kanal
05638  554d  20 eb 90          jsr coout        O-Kanal öffnen/listen, Fehlertest
                                                (BASIC)
05639  5550  28                plp
05640  5551
05641  5551  4c 5a 55          jmp print        BASIC-Statement PRINT ($99)
05642  5554
05643  5554
05644  5554  ===> PRINT String <===
05645  5554
05646  5554  20 e5 55   strdon jsr strprt       Ausgabe Arbeitsstring auf akt. Kanal
05647  5557
05648  5557
05649  5557  ===> Parameter nächster Ausdruck PRINT <===
05650  5557
05651  5557  20 86 03   newchr jsr chrgot       letztes Zeichen erneut nach ac
05652  555a
05653  555a
05654  555a  ===> BASIC-Statement PRINT ($99) <===
05655  555a
05656  555a  f0 3c       print  beq crdo         Ausgabe cr (+lf) auf akt. Kanal
05657  555c
05658  555c  c9 fb             cmp #_usint      USING Token
05659  555e  d0 03             bne printc
05660  5560
05661  5560  4c 20 95          jmp using
05662  5563
05663  5563
05664  5563  f0 43       printc beq prtrts
05665  5565
05666  5565  c9 a3             cmp #_tabt       TAB( Token
05667  5567  f0 50             beq taber        Tabulierung ausführen
05668  5569
05669  5569  c9 a6             cmp #_spct       SPC( Token
05670  556b  18                clc
05671  556c  f0 4b             beq taber        Tabulierung ausführen
05672  556e
05673  556e  c9 2c             cmp #','         10-er Tabulierung
05674  5570  f0 37             beq comprt       10-er Tabulierung bei PRINT <Komma>
05675  5572
05676  5572  c9 3b             cmp #';'
```

```
05677  5574  f0 5e            beq notabr       chrget, Test weiter PRINT
05678  5576
05679  5576  20 ef 77         jsr frmevl       folgenden Ausdruck auswerten
05680  5579
05681  5579  24 0f            bit valtyp       Flag Variablentyp
05682  557b  30 d7            bmi strdon       PRINT String
05683  557d
05684  557d  ===> PRINT numerischen Ausdruck <===
05685  557d
05686  557d  20 42 8e         jsr fout         FAC nach String, Adr. nach ac/yr
05687  5580  20 9a 86         jsr strlit       String (Descr.ac/yr) nach Arb.string
05688  5583  20 e5 55         jsr strprt       Ausgabe Arbeitsstring auf akt. Kanal
05689  5586  20 00 56         jsr outspc       Ausgabe _CRITE, wenn BS, sonst _SPC
05690  5589  d0 cc            bne newchr       immer
05691  558b
05692  558b
05693  558b  ===> Abschluß Zeilen-Eingabe <===
05694  558b
05695  558b  a9 00      fininl lda #_eom       end of message
05696  558d  9d 00 02         sta buf,x        Eingabe-Puffer
05697  5590
05698  5590  a2 ff            ldx #<buf-1      Eingabe-Puffer
05699  5592  a0 01            ldy #>buf-1      Eingabe-Puffer
05700  5594  a5 15            lda channl       aktiver I/O-Kanal
05701  5596  d0 10            bne prtrts       nicht Tastatur
05702  5598
05703  5598
05704  5598  ===> Ausgabe cr (+lf) auf akt. Kanal <===
05705  5598
05706  5598  a9 0d      crdo   lda #_cr         carriage return
05707  559a  20 0c 56         jsr outdo        Ausgabe ac auf akt. Kanal
05708  559d
05709  559d  24 15            bit channl       aktiver I/O-Kanal
05710  559f  10 05            bpl crfin
05711  55a1
05712  55a1  a9 0a            lda #_lf         line feed
05713  55a3  20 0c 56         jsr outdo        Ausgabe ac auf akt. Kanal
05714  55a6
05715  55a6  49 ff      crfin  eor #%11111111
05716  55a8  60         prtrts rts
05717  55a9
05718  55a9
05719  55a9  ===> 10-er Tabulierung bei PRINT <Komma> <===
05720  55a9
05721  55a9  38        comprt sec
05722  55aa  20 8d 92         jsr cplot        Cursor-Pos. (rel. BS-Fenster)
                                               les./schreib.
05723  55ad  98               tya
05724  55ae
05725  55ae  38               sec
05726  55af  e9 0a      morco1 sbc #10
05727  55b1  b0 fc            bcs morco1
05728  55b3
05729  55b3  49 ff            eor #$ff
05730  55b5  69 01            adc #1
05731  55b7  d0 16            bne aspac        Funktion TAB( ($a3)
05732  55b9
05733  55b9
```

```
05734  55b9
05734  55b9  ===> Tabulierung ausführen <===
05735  55b9
05736  55b9  08          taber  php
05737  55ba  38                 sec
05738  55bb  20 8d 92           jsr cplot        Cursor-Pos. (rel. BS-Fenster)
                                                 les./schreib.
05739  55be  84 0b              sty trmpos       Tabulator-Spalte
05740  55c0
05741  55c0  20 f1 87           jsr gtbytc       chrget, 8 Bit INT nach xr
05742  55c3  c9 29              cmp #')'
05743  55c5  d0 13              bne snerr4       Behandlung »?syntax error«
05744  55c7
05745  55c7  28                 plp
05746  55c8  90 06              bcc xspac        Funktion SPC( ($a6)
05747  55ca
05748  55ca  8a                 txa
05749  55cb  e5 0b              sbc trmpos       Tabulator-Spalte
05750  55cd  90 05              bcc notabr       chrget, Test weiter PRINT
05751  55cf
05752  55cf
05753  55cf  ===> Funktion TAB( ($a3) <===
05754  55cf
05755  55cf  aa          aspac  tax
05756  55d0
05757  55d0
05758  55d0  ===> Funktion SPC( ($a6) <===
05759  55d0
05760  55d0  e8          xspac  inx
05761  55d1  ca          xspac2 dex
05762  55d2  d0 09              bne xspac1
05763  55d4
05764  55d4
05765  55d4  ===> chrget, Test weiter PRINT <===
05766  55d4
05767  55d4  20 80 03    notabr jsr chrget       nächstes Zeichen nach ac
05768  55d7  4c 63 55           jmp printc
05769  55da
05770  55da
05771  55da  ===> Behandlung »?syntax error« <===
05772  55da
05773  55da  4c 6c 79    snerr4 jmp snerr        Behandlung »?syntax error«
05774  55dd
05775  55dd
05776  55dd  20 00 56    xspac1 jsr outspc       Ausgabe _CRITE, wenn BS, sonst _SPC
05777  55e0  d0 ef              bne xspac2       immer
05778  55e2
05779  55e2
05780  55e2  ===> Ausgabe String ac/yr <===
05781  55e2
05782  55e2  20 9a 86    strout jsr strlit       String (Descr.ac/yr) nach Arb.string
05783  55e5
05784  55e5
05785  55e5  ===> Ausgabe Arbeitsstring auf akt. Kanal <===
05786  55e5
05787  55e5  20 81 87    strprt jsr frefac       Daten Arbeitsstring aus Ablage,
                                                 Adresse index1, Länge ac
05788  55e8
```

```
05789  55e8  aa              tax
05790  55e9  a0 00           ldy #0
05791  55eb  e8              inx
05792  55ec  ca        strpr2 dex
05793  55ed  f0 b9           beq prtrts
05794  55ef
05795  55ef  20 b7 03        jsr ind1_1      LDA (index1),y aus VAR-RAM
05796  55f2  20 0c 56        jsr outdo       Ausgabe ac auf akt. Kanal
05797  55f5  c8              iny
05798  55f6  c9 0d           cmp #_cr        carriage return
05799  55f8  d0 f2           bne strpr2
05800  55fa
05801  55fa  20 a6 55        jsr crfin
05802  55fd  4c ec 55        jmp strpr2
05803  5600
05804  5600
05805  5600  ===> Ausgabe _CRITE, wenn BS, sonst _SPC <===
05806  5600
05807  5600  a5 15     outspc lda channl      aktiver I/O-Kanal
05808  5602  f0 03           beq crtskp      Ausgabe _CRITE akt. Kanal
05809  5604
05810  5604
05811  5604  ===> Ausgabe _SPC akt. Kanal <===
05812  5604
05813  5604  a9 20     realsp lda #_spc
05814  5606  2c              .byte _skip2
05815  5607
05816  5607
05817  5607  ===> Ausgabe _CRITE akt. Kanal <===
05818  5607
05819  5607  a9 1d     crtskp lda #_crite     cursor rechts
05820  5609  2c              .byte _skip2
05821  560a
05822  560a
05823  560a  ===> Ausgabe Fragez. akt. Kanal <===
05824  560a
05825  560a  a9 3f     outqst lda #'?'
05826  560c
05827  560c
05828  560c  ===> Ausgabe ac auf akt. Kanal <===
05829  560c
05830  560c  20 df 90   outdo  jsr outch       Ausgabe ac akt. Kanal, Fehlertest
                                              (BASIC)
05831  560f  29 ff           and #%11111111
05832  5611  60              rts
05833  5612
05834  5612
05835  5612  ===> BASIC-Statement GET ($a1) <===
05836  5612
05837  5612  20 d9 84   get    jsr errdir      Test Direktmodus, Behandl. »?illegal
                                              direct error«
05838  5615  85 77           sta tmptk       Token für inpco1
05839  5617
05840  5617  c9 23           cmp #'#'
05841  5619  f0 0a           beq getn        GET# aus Datei
05842  561b
05843  561b  c9 f9           cmp #_keyt      KEY Token
```

```
05844  561d  d0 16           bne gettty      BASIC-Statement GET ($a1), GETKEY
                                             ($a1 $f9)
05845  561f
05846  561f  20 80 03        jsr chrget      nächstes Zeichen nach ac
05847  5622  4c 35 56        jmp gettty      BASIC-Statement GET ($a1), GETKEY
                                             ($a1 $f9)
05848  5625
05849  5625
05850  5625  ===> GET# aus Datei <===
05851  5625
05852  5625  20 80 03  getn   jsr chrget      nächstes Zeichen nach ac
05853  5628  20 f4 87        jsr getbyt      8 Bit INT üb. frmnum nach xr
05854  562b
05855  562b  a9 2c           lda #','
05856  562d  20 5e 79        jsr synchr      Test auf Zeichen in ac, sonst Fehler
05857  5630
05858  5630  86 15           stx channl      aktiver I/O-Kanal
05859  5632  20 fd 90        jsr coin        I-Kanal öffnen, Fehlertest (BASIC)
05860  5635
05861  5635
05862  5635  ===> BASIC-Statement GET ($a1), GETKEY ($a1 $f9) <===
05863  5635
05864  5635  a2 01      gettty ldx #<buf+1    Eingabe-Puffer
05865  5637  a0 02             ldy #>buf+1    Eingabe-Puffer
05866  5639
05867  5639  a9 00           lda #0
05868  563b  8d 01 02        sta buf+1       Eingabe-Puffer +1
05869  563e
05870  563e  a9 40           lda #_get       GET Flag
05871  5640  20 b2 56        jsr inpco1
05872  5643
05873  5643  a6 15           ldx channl      aktiver I/O-Kanal
05874  5645  d0 13           bne iorele      in ac Kanal clr, Tastatur aktivieren
05875  5647  60              rts
05876  5648
05877  5648
05878  5648  ===> BASIC-Statement INPUT# ($84) <===
05879  5648
05880  5648  20 f4 87   inputn jsr getbyt     la nach xr
05881  564b
05882  564b  a9 2c           lda #','
05883  564d  20 5e 79        jsr synchr      Test auf Zeichen in ac, sonst Fehler
05884  5650
05885  5650  86 15           stx channl      aktiver I/O-Kanal
05886  5652
05887  5652  20 fd 90        jsr coin        I-Kanal öffnen, Fehlertest (BASIC)
05888  5655  20 71 56        jsr notqti      INPUT ohne Kommentar
05889  5658
05890  5658
05891  5658  ===> akt. Kanal clr, Tastatur aktivieren <===
05892  5658
05893  5658  a5 15      iodone lda channl     aktiver I/O-Kanal
05894  565a
05895  565a
05896  565a  ===> in ac Kanal clr, Tastatur aktivieren <===
05897  565a
05898  565a  20 6f 92   iorele jsr cclrch     ind. I/O-Kanäle schließen
05899  565d
```

```
05900  565d  a2 00            ldx #_keybd      Tastatur
05901  565f  86 15            stx channl       aktiver I/O-Kanal
05902  5661
05903  5661  60               rts
05904  5662
05905  5662
05906  5662  ===> BASIC-Statement INPUT ($85) <===
05907  5662
05908  5662  c9 22      input  cmp #'"'
05909  5664  d0 0b             bne notqti       INPUT ohne Kommentar
05910  5666
05911  5666  20 13 79          jsr strtxt       String aus PGM nach Arb.string
05912  5669
05913  5669  a9 3b             lda #';'
05914  566b  20 5e 79          jsr synchr       Test auf Zeichen in ac, sonst Fehler
05915  566e
05916  566e  20 e5 55          jsr strprt       Ausgabe Arbeitsstring auf akt. Kanal
05917  5671
05918  5671
05919  5671  ===> INPUT ohne Kommentar <===
05920  5671
05921  5671  20 d9 84   notqti jsr errdir       Test Direktmodus, Behandl. »?illegal
                                                direct error«
05922  5674
05923  5674  a9 2c             lda #','
05924  5676  8d ff 01          sta buf-1        vor Eingabepuffer
05925  5679
05926  5679  20 9c 56   getagn jsr qinlin       Ausg. Fragez., Eingabe  in Puffer
05927  567c
05928  567c  a5 15             lda channl       aktiver I/O-Kanal
05929  567e  f0 0d             beq bufful       Tastatur
05930  5680
05931  5680  20 51 92          jsr creads       RS-/System-Status nach ac
05932  5683  29 02             and #%00000010
05933  5685
05934  5685   ......1.  1 = Timeout Lesen
05935  5685
05936  5685  f0 06             beq bufful
05937  5687
05938  5687  ===> Timeout Lesen <===
05939  5687
05940  5687  20 58 56          jsr iodone       akt. Kanal clr, Tastatur aktivieren
05941  568a  4c 8f 52          jmp data         BASIC-Statement DATA ($83)
05942  568d
05943  568d
05944  568d  ad 00 02   bufful lda buf          Eingabe-Puffer
05945  5690  d0 1e             bne inpcon       1. Zeichen nicht _eom
05946  5692
05947  5692  a5 15             lda channl       aktiver I/O-Kanal
05948  5694  d0 e3             bne getagn       nicht Tastatur
05949  5696
05950  5696  20 a2 52          jsr datan        yr hinter nächst. »:« bzw. $00
05951  5699  4c 92 52          jmp addon        yr zu txtptr addieren
05952  569c
05953  569c
05954  569c  ===> Ausg. Fragez., Eingabe  in Puffer <===
05955  569c
05956  569c  a5 15      qinlin lda channl       aktiver I/O-Kanal
```

```
05957  569e  d0 06            bne ginlin       Eingabe in Puffer
05958  56a0
05959  56a0  20 0a 56         jsr outqst       Ausgabe Fragez. akt. Kanal
05960  56a3  20 04 56         jsr realsp       Ausgabe _SPC akt. Kanal
05961  56a6  4c 93 4f  ginlin jmp inlin        Eingabe in Puffer
05962  56a9
05963  56a9
05964  56a9  ===> BASIC-Statement READ ($87) <===
05965  56a9
05966  56a9  a6 43      read   ldx datptr      ADDRL Zeiger auf DATA (READ)
05967  56ab  a4 44             ldy datptr+1    ADDRH Zeiger auf DATA (READ)
05968  56ad
05969  56ad  ===> DATA Konstante in Var. einlesen <===
05970  56ad
05971  56ad  a9 98             lda #_read      READ Flag
05972  56af  2c                .byte _skip2
05973  56b0
05974  56b0  ===> INPUT in Var. einlesen <===
05975  56b0
05976  56b0  a9 00      inpcon lda #_input     INPUT Flag
05977  56b2  85 13      inpco1 sta inpflg      Flag Herkunft Aufruf inloop
05978  56b4
05979  56b4  86 45             stx inpptr      ADDRL Zeiger auf Input-Quelle
05980  56b6  84 46             sty inpptr+1    ADDRH Zeiger auf Input-Quelle
05981  56b8
05982  56b8  20 af 7a   inloop jsr ptrget      folg. Variable suchen/anlegen
05983  56bb  85 4b             sta lstpnt      ADDRL Zeiger auf Var.
05984  56bd  84 4c             sty lstpnt+1    ADDRH Zeiger auf Var.
05985  56bf
05986  56bf  a2 01             ldx #1
05987  56c1  b5 3d      inlop0 lda txtptr,x    ADDRL chrget-Zeiger
05988  56c3  95 4d             sta vartxt,x    ADDRL Zeiger VAR in PGM
05989  56c5
05990  56c5  b5 45             lda inpptr,x    ADDRL Zeiger auf Input-Quelle
05991  56c7  95 3d             sta txtptr,x    ADDRL chrget-Zeiger
05992  56c9
05993  56c9  ca                dex
05994  56ca  10 f5             bpl inlop0
05995  56cc
05996  56cc  20 86 03          jsr chrgot      letztes Zeichen erneut nach ac
05997  56cf  d0 31             bne datbk1      DATA nach Var.
05998  56d1
05999  56d1  24 13             bit inpflg      Flag Herkunft Aufruf inloop
06000  56d3
06001  56d3   .6.....  1 = get
06002  56d3
06003  56d3  50 1a             bvc qdata       Test INPUT/READ
06004  56d5
06005  56d5  ===> Behandlung GET <===
06006  56d5
06007  56d5  a5 77             lda tmptk       Zeichen nach GET im Aufruf
06008  56d7  c9 f9             cmp #_keyt      KEY Token (GETKEY)
06009  56d9  d0 08             bne inlop1      Behandlung GET, GET#
06010  56db
06011  56db  ===> Behandlung GETKEY <===
06012  56db
06013  56db
```

```
06014  56db  ===> Warten auf Tast.eing. GETKEY <===
06015  56db
06016  56db  20 09 91   qdat1  jsr cgetl        Tastaturpuffer nach ac
06017  56de  aa                tax              nach xr
06018  56df  f0 fa              beq qdat1        leer, warten
06019  56e1  d0 03              bne inlop2       immer
06020  56e3
06021  56e3
06022  56e3  ===> Behandlung GET, GET# <===
06023  56e3
06024  56e3  20 09 91   inlop1 jsr cgetl        ein Zeichen nach ac
06025  56e6  8d 00 02   inlop2 sta buf          Zeichen in Eingabepuffer
06026  56e9
06027  56e9  a2 ff              ldx #<buf-1      ADDRL vor Eingabepuff.
06028  56eb  a0 01              ldy #>buf-1      ADDRH vor Eingabepuff.
06029  56ed  d0 0f              bne datbk        immer
06030  56ef
06031  56ef
06032  56ef  ===> Test INPUT/READ <===
06033  56ef
06034  56ef  10 03      qdata  bpl qdata1       INPUT
06035  56f1
06036  56f1  ===> Behandlung READ <===
06037  56f1
06038  56f1  4c ca 57           jmp datlop       nächstes DATA suchen
06039  56f4
06040  56f4
06041  56f4  ===> Behandlung INPUT <===
06042  56f4
06043  56f4  a5 15      qdata1 lda channl       aktiver I/O-Kanal
06044  56f6  d0 03             bne getnth
06045  56f8
06046  56f8  ===> INPUT von Tastatur <===
06047  56f8
06048  56f8  20 0a 56           jsr outqst       Ausgabe Fragez. akt. Kanal
06049  56fb
06050  56fb  20 9c 56   getnth jsr qinlin       Ausg. Fragez., Eingabe  in Puffer
06051  56fe
06052  56fe  86 3d      datbk  stx txtptr       ADDRL Zeiger vor Eingabe
06053  5700  84 3e             sty txtptr+1     ADDRH Zeiger vor Eingabe
06054  5702
06055  5702
06056  5702  ===> DATA nach Var. <===
06057  5702
06058  5702  20 80 03   datbk1 jsr chrget       Zeichen (Puffer) nach ac
06059  5705
06060  5705  24 0f             bit valtyp       Flag Variablentyp
06061  5707  10 31             bpl numins       Eingabe in num. Var.
06062  5709
06063  5709  ===> Eingabe in String Var. <===
06064  5709
06065  5709  24 13             bit inpflg       Flag Herkunft Aufruf inloop
06066  570b
06067  570b   .6.....  1 = get
06068  570b
06069  570b  50 09             bvc setqut       Test Abbruchzeichen in Eingabe
06070  570d
```

```
06071  570d  ===> Behandlung GET <===
06072  570d
06073  570d  e8              inx             Zeiger auf _eom
06074  570e  86 3d           stx txtptr      ADDRL chrget-Zeiger
06075  5710
06076  5710  a9 00           lda #_eom
06077  5712  85 09           sta charac      Suchzeichen
06078  5714  f0 0c           beq resetc      immer
06079  5716
06080  5716
06081  5716  ===> Test Abbruchzeichen in Eingabe <===
06082  5716
06083  5716  85 09     setqut sta charac     Suchzeichen
06084  5718
06085  5718  c9 22           cmp #'"'        Abbruchzeichen
06086  571a  f0 07           beq nowget
06087  571c
06088  571c  a9 3a           lda #':'        Abbruchzeichen
06089  571e  85 09           sta charac      Suchzeichen
06090  5720
06091  5720  a9 2c           lda #','        Abbruchzeichen
06092  5722
06093  5722  18        resetc clc
06094  5723  85 0a     nowget sta endchr     Abbruchzeichen
06095  5725
06096  5725  a5 3d           lda txtptr      ADDRL chrget-Zeiger
06097  5727  a4 3e           ldy txtptr+1    ADDRH chrget-Zeiger
06098  5729
06099  5729  69 00           adc #0
06100  572b  90 01           bcc nowge1
06101  572d  c8              iny
06102  572e
06103  572e  20 a0 86  nowge1 jsr strlt2      String übertragen
06104  5731  20 1f 79        jsr st2txt      txtptr hinter String in PGM
06105  5734  20 05 54        jsr inpcom      String zuweisen
06106  5737  4c 44 57        jmp strdn2      Test auf richt. Trennzeichen
06107  573a
06108  573a
06109  573a  ===> Eingabe in num. Var. <===
06110  573a
06111  573a  a2 00     numins ldx #_pgmbk     PGM Bank
06112  573c  20 22 8d        jsr fin         Zahlenstr. (Bnk.=xr) nach FAC
06113  573f
06114  573f  a5 10           lda intflg      Flag für Gleitkomma/Integer
06115  5741  20 e3 53        jsr qintgr      GK/INT aus FAC nach Var.
06116  5744
06117  5744
06118  5744  ===> Test auf richt. Trennzeichen <===
06119  5744
06120  5744  20 86 03  strdn2 jsr chrgot      letztes Zeichen erneut nach ac
06121  5747  f0 3b           beq trmok       Trennzeichen
06122  5749
06123  5749  c9 2c           cmp #','
06124  574b  f0 37           beq trmok       txtptr wiederherstellen
06125  574d
06126  574d  a5 13           lda inpflg      Flag Herkunft Aufruf inloop
06127  574f
06128  574f   00000000      input
```

```
06129  574f   7.......  1 = read
06130  574f
06131  574f  f0 0a             beq trmno1
06132  5751  30 04             bmi tmerr        Behandlung »?type mismatch errror«
06133  5753
06134  5753  a6 15             ldx channl       aktiver I/O-Kanal
06135  5755  d0 08             bne fiderr       Behandlung »?file data error«
06136  5757
06137  5757
06138  5757
06138  5757  ===> Behandlung »?type mismatch errror« <===
06139  5757
06140  5757  a2 16       tmerr  ldx #_ertm
06141  5759  d0 06              bne jerro1       ind. Behandlung Fehlermeldung
06142  575b
06143  575b  a5 15       trmno1 lda channl       aktiver I/O-Kanal
06144  575d  f0 05              beq doagin       Behandlung »?redo from start«
06145  575f
06146  575f
06147  575f  ===> Behandlung »?file data error« <===
06148  575f
06149  575f  a2 18       fiderr ldx #_erbd
06150  5761
06151  5761  ===> ind. Behandlung Fehlermeldung <===
06152  5761
06153  5761  4c 3c 4d    jerro1 jmp error        ind. Behandlung Fehlermeldung
06154  5764
06155  5764
06156  5764  ===> Behandlung »?redo from start« <===
06157  5764
06158  5764  20 81 92    doagin jsr cprimm       folg. Text ausg./überspringen
06159  5767
06160  5767  3f 52 45           .byte »?redo from start«, _cr, _eom
06160  576a  44 4f 20
06160  576d  46 52 4f
06160  5770  4d 20 53
06160  5773  54 41 52
06160  5776  54 0d 00
06161  5779
06162  5779  ad 02 12           lda oldtxt       ADDRL Zeiger letzter BASIC-Befehl
06163  577c  ac 03 12           ldy oldtxt+1     ADDRH Zeiger letzter BASIC-Befehl
06164  577f
06165  577f  85 3d              sta txtptr       ADDRL chrget-Zeiger
06166  5781  84 3e              sty txtptr+1     ADDRH chrget-Zeiger
06167  5783
06168  5783  60                 rts
06169  5784
06170  5784
06171  5784  ===> txtptr wiederherstellen <===
06172  5784
06173  5784  a2 01       trmok  ldx #1
06174  5786
06175  5786  b5 3d       trmlp  lda txtptr,x     ADDRL chrget-Zeiger
06176  5788  95 45              sta inpptr,x     ADDRL Zeiger auf Input-Quelle
06177  578a
06178  578a  b5 4d              lda vartxt,x     ADDRL Zeiger VAR in PGM
06179  578c  95 3d              sta txtptr,x     ADDRL chrget-Zeiger
06180  578e
```

```
06181  578e  ca              dex
06182  578f  10 f5           bpl trmlp
06183  5791
06184  5791  ===> Test auf weitere Var. <===
06185  5791
06186  5791  20 86 03        jsr chrgot      letztes Zeichen erneut nach ac
06187  5794  f0 06           beq endvar      Trennzeichen
06188  5796
06189  5796
06189  5796  ===> Behandlung weitere Var. <===
06190  5796
06191  5796  20 5c 79        jsr chkcom      Test auf folg. »Komma«, sonst Fehler
06192  5799  4c b8 56        jmp inloop
06193  579c
06194  579c
06195  579c  ===> Abschluß Einlesen in Var. <===
06196  579c
06197  579c  a5 45     endvar lda inpptr      ADDRL Zeiger auf Input-Quelle
06198  579e  a4 46            ldy inpptr+1    ADDRH Zeiger auf Input-Quelle
06199  57a0
06200  57a0  a6 13            ldx inpflg      Flag Herkunft Aufruf inloop
06201  57a2
06202  57a2   7.......  1 = read
06203  57a2
06204  57a2  10 05            bpl endv10
06205  57a4
06206  57a4  ===> Zeiger in DATAs speichern <===
06207  57a4
06208  57a4  85 43            sta datptr      ADDRL Zeiger auf DATA (READ)
06209  57a6  84 44            sty datptr+1    ADDRH Zeiger auf DATA (READ)
06210  57a8  60               rts
06211  57a9
06212  57a9
06213  57a9  a0 00     endv10 ldy #0
06214  57ab  a9 45            lda #<inpptr    ADDRL Zeiger auf Input-Quelle
06215  57ad  20 9f 03         jsr iram0       Zeichen hinter Eingabe
06216  57b0  f0 17            beq inprts      _eom
06217  57b2
06218  57b2  a5 15            lda channl      aktiver I/O-Kanal
06219  57b4  d0 13            bne inprts      nicht Tastatur
06220  57b6
06221  57b6  ===> Ausgabe »?extra ignored« <===
06222  57b6
06223  57b6  20 81 92         jsr cprimm      folg. Text ausg./überspringen
06224  57b9
06225  57b9  3f 45 58         .byte '»?extra ignored', _cr, _eom
06225  57bc  54 52 41
06225  57bf  20 49 47
06225  57c2  4e 4f 52
06225  57c5  45 44 0d
06225  57c8  00
06226  57c9
06227  57c9  60        inprts rts
06228  57ca
06229  57ca
06230  57ca  ===> nächstes DATA suchen <===
06231  57ca
06232  57ca  20 a2 52  datlop jsr datan       yr hinter nächst. »:« bzw. $00
```

```
06233  57cd  c8              iny             Zeiger hinter Trennzeichen
06234  57ce
06235  57ce  aa              tax             Trennzeichen
06236  57cf  d0 15           bne nowlin      nicht Zeilenende
06237  57d1
06238  57d1  ===> Test PGM Ende <===
06239  57d1
06240  57d1  a2 0d           ldx #_erod      vorbesetzen »?out of data error«
06241  57d3
06242  57d3  c8              iny             ADDRH Linkadresse
06243  57d4  20 c9 03        jsr indtxt      LDA (txtptr),y aus PGM-RAM
06244  57d7  f0 40           beq jerro2      ind. Behandlung Fehlermeldung
06245  57d9
06246  57d9  ===> akt. Zeilennr. nach datlin <===
06247  57d9
06248  57d9  c8              iny             Low-Byte Zeilennr.
06249  57da  20 c9 03        jsr indtxt      LDA (txtptr),y aus PGM-RAM
06250  57dd  85 41           sta datlin      ADDRL DATA-Zeilennummer
06251  57df
06252  57df  c8              iny             High-Byte Zeilennr.
06253  57e0  20 c9 03        jsr indtxt      LDA (txtptr),y aus PGM-RAM
06254  57e3
06255  57e3  c8              iny             erstes Zeichen akt. Zeile
06256  57e4  85 42           sta datlin+1    ADDRH DATA-Zeilennummer
06257  57e6
06258  57e6  20 92 52   nowlin jsr addon      yr zu txtptr addieren
06259  57e9  20 86 03        jsr chrgot      letztes Zeichen erneut nach ac
06260  57ec
06261  57ec  aa              tax
06262  57ed  e0 83           cpx #_datat     DATA Token
06263  57ef  d0 d9           bne datlop      weitersuchen
06264  57f1
06265  57f1  4c 02 57        jmp datbk1      DATA nach Var.
06266  57f4
06267  57f4
06268  57f4                  .end
06269  57f4                  .lib basic.cmd_3
06270  57f4  put"@basic.cmd_3"
```

```
06272  57f4
06273  57f4
06274  57f4  ===> BASIC-Statement NEXT ($82) <===
06275  57f4
06276  57f4  d0 13      next   bne next10       NEXT mit Parameter
06277  57f6
06278  57f6  ===> Trennzeichen, NEXT ohne Parameter <===
06279  57f6
06280  57f6  a0 ff             ldy #$ff
06281  57f8  d0 14             bne next20       immer
06282  57fa
06283  57fa
06284  57fa  ===> Test weitere NEXT Parameter <===
06285  57fa
06286  57fa  a0 12      next50 ldy #_lnfor      Stapelbytes FOR
06287  57fc  20 59 50          jsr rlsstk       BASIC-Stapel Zeiger um yr erhöhen
06288  57ff
06289  57ff  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
06290  5802  c9 2c             cmp #','
06291  5804  d0 71             bne next45
06292  5806
06293  5806  20 80 03          jsr chrget       nächstes Zeichen nach ac
06294  5809
06295  5809
06296  5809  ===> NEXT mit Parameter <===
06297  5809
06298  5809  20 af 7a   next10 jsr ptrget       folg. Variable suchen/anlegen
06299  580c
06300  580c  85 4b             sta lstpnt
06301  580e  84 4c      next20 sty lstpnt+1
06302  5810
06303  5810  a9 81             lda #_fort       FOR Token
06304  5812  20 aa 4f          jsr search       Token (ac) in BASIC-Stapel suchen
06305  5815  f0 05             beq next30
06306  5817
06307  5817  ===> Behandlung »?next without for error« <===
06308  5817
06309  5817  a2 0a             ldx #_ernwf
06310  5819
06311  5819  ===> ind. Behandlung Fehlermeldung <===
06312  5819
06313  5819  4c 3c 4d   jerro2 jmp error       ind. Behandlung Fehlermeldung
06314  581c
06315  581c
06316  581c  20 50 50   next30 jsr movfnd      Kopie Suchzeiger nach Stapelzeiger
06317  581f
06318  581f  a5 3f             lda fndpnt       ADDRL BASIC-Stapel Suchzeiger
06319  5821  18                clc
06320  5822  69 03             adc #_step       ac/yr auf STEP Wert
06321  5824
06322  5824  a4 40             ldy fndpnt+1     ADDRH BASIC-Stapel Suchzeiger
06323  5826  90 01             bcc next35
06324  5828  c8                iny
06325  5829
06326  5829  20 d4 8b   next35 jsr movfm        STEP Wert nach FAC
06327  582c
06328  582c  a0 08             ldy #_stpsg      Zeiger auf Vorzeichen STEP Wert
06329  582e  b1 3f             lda (fndpnt),y   Vorzeichen STEP Wert
```

```
06330  5830  85 68           sta facsgn       FAC: SG
06331  5832
06332  5832  a0 01           ldy #_fords      Zeiger ADDRL Desc. FOR Var.
06333  5834  b1 3f           lda (fndpnt),y   ADDRL BASIC-Stapel Suchzeiger
06334  5836
06335  5836  48              pha              ADDRL Desc. FOR Var.
06336  5837  aa              tax              ADDRL Desc. FOR Var.
06337  5838
06338  5838  c8              iny
06339  5839  b1 3f           lda (fndpnt),y   ADDRL BASIC-Stapel Suchzeiger
06340  583b
06341  583b  48              pha              ADDRH Desc. FOR Var.
06342  583c
06343  583c  a8              tay              ADDRH Desc. FOR Var.
06344  583d  8a              txa              ADDRL Desc. FOR Var.
06345  583e  20 45 88        jsr fadd         FOR plus STEP nach FAC
06346  5841
06347  5841  68              pla
06348  5842  a8              tay              ADDRH Desc. FOR Var.
06349  5843
06350  5843  68              pla
06351  5844  aa              tax              ADDRL Desc. FOR Var.
06352  5845
06353  5845  8d 04 ff        sta lcrd         ROM / VAR-RAM einschalten
06354  5848  20 00 8c        jsr movmf        FAC runden, nach FOR Var.
06355  584b
06356  584b  ===> Adresse Endwert FOR nach ac/yr <===
06357  584b
06358  584b  a5 3f           lda fndpnt       ADDRL BASIC-Stapel Suchzeiger
06359  584d  18              clc
06360  584e  69 09           adc #9           ac
06361  5850
06362  5850  a4 40           ldy fndpnt+1     ADDRH BASIC-Stapel Suchzeiger
06363  5852  90 01           bcc next40
06364  5854  c8              iny              yr
06365  5855
06366  5855  8d 03 ff  next40 sta lcrc        ROM / PGM-RAM einschalten
06367  5858  20 87 8c        jsr fcomp        FAC mit Var. ac/yr vergleichen
06368  585b
06369  585b  a0 08           ldy #8
06370  585d  38              sec
06371  585e  f1 3f           sbc (fndpnt),y   ADDRL BASIC-Stapel Suchzeiger
06372  5860  f0 98           beq next50       Test weitere NEXT Parameter
06373  5862
06374  5862  ===> FOR Schleife starten <===
06375  5862
06376  5862  a0 11           ldy #17
06377  5864  b1 3f           lda (fndpnt),y   ADDRL Schleifen Anfang
06378  5866  85 3d           sta txtptr       ADDRL chrget-Zeiger
06379  5868
06380  5868  88              dey
06381  5869  b1 3f           lda (fndpnt),y   ADDRH Schleifen Anfang
06382  586b  85 3e           sta txtptr+1     ADDRH chrget-Zeiger
06383  586d
06384  586d  88              dey
06385  586e  b1 3f           lda (fndpnt),y   Low-Byte Zeilennr.
06386  5870  85 3c           sta curlin+1     High-Byte akt. Zeilennummer
06387  5872
```

```
06388  5872  88                dey
06389  5873  b1 3f             lda (fndpnt),y   High-Byte Zeilennr.
06390  5875  85 3b             sta curlin       Low-Byte akt. Zeilennummer
06391  5877
06392  5877  60          next45 rts
06393  5878
06394  5878
06395  5878  ===> weitere Felddimensionierung <===
06396  5878
06397  5878  20 5c 79    dim3   jsr chkcom      Test auf folg. »Komma«, sonst Fehler
06398  587b
06399  587b
06400  587b  ===> BASIC-Statement DIM ($86) <===
06401  587b
06402  587b  aa          dim    tax             DIM Flag <> 0
06403  587c  20 b4 7a            jsr ptrgt1      folg. VAR dim. / anlegen / Zeiger
                                                such.
06404  587f
06405  587f  ===> Test weitere Felddimensionierung <===
06406  587f
06407  587f  20 86 03            jsr chrgot      letztes Zeichen erneut nach ac
06408  5882  d0 f4               bne dim3        weitere Felddimensionierung
06409  5884
06410  5884  60                  rts
06411  5885
06412  5885
06413  5885  ===> BASIC-Statement SYS ($9e) <===
06414  5885
06415  5885  20 12 88    sys    jsr getwrd      PC Adresse nach linnum
06416  5888
06417  5888  a5 16               lda linnum      ADDRL PC
06418  588a  85 04               sta pclo        Puffer 8502 CPU ADDRL PC
06419  588c
06420  588c  a5 17               lda linnum+1    ADDRH PC
06421  588e  85 03               sta pchi        Puffer 8502 CPU ADDRH PC
06422  5890
06423  5890  ad d5 03            lda curbnk      akt. Bank
06424  5893  85 02               sta bnk         Bank Nummer
06425  5895
06426  5895  ===> CPU-Register einlesen <===
06427  5895
06428  5895  20 1e 9e            jsr optbyt      Test Angabe ac
06429  5898  90 02               bcc sys10       nur Komma
06430  589a  86 06               stx areg        Puffer 8502 CPU AC
06431  589c
06432  589c  20 1e 9e    sys10   jsr optbyt      Test Angabe xr
06433  589f  90 02               bcc sys20       nur Komma
06434  58a1  86 07               stx xreg        Puffer 8502 CPU XR
06435  58a3
06436  58a3  20 1e 9e    sys20   jsr optbyt      Test Angabe yr
06437  58a6  90 02               bcc sys30       nur Komma
06438  58a8  86 08               stx yreg        Puffer 8502 CPU YR
06439  58aa
06440  58aa  20 1e 9e    sys30   jsr optbyt      Test Angabe sr
06441  58ad  90 02               bcc sys40       nur Komma
06442  58af  86 05               stx sreg        Puffer 8502 CPU SR
06443  58b1
```

```
06444  58b1  4c 6e ff    sys40  jmp kjsrfr        JSR in andere Bank, RTS in akt. Bank
                                                  (Sprungv.)
06445  58b4
06446  58b4
06447  58b4  ===> BASIC-Statement TRON ($d8) <===
06448  58b4
06449  58b4  a9 ff       tron   lda #$ff
06450  58b6  2c                 .byte _skip2
06451  58b7
06452  58b7
06453  58b7  ===> BASIC-Statement TROFF ($d9) <===
06454  58b7
06455  58b7  a9 00       troff  lda #0
06456  58b9  8d 6f 11           sta trcflg        TRACE-Flag (0=aus, ff=ein)
06457  58bc  60                 rts
06458  58bd
06459  58bd
06460  58bd  ===> BASIC-Statement RREG ($fe $09) <===
06461  58bd
06462  58bd  a9 00       rreg   lda #0
06463  58bf  85 0d              sta count         Zähl./Zeig. Puffer CPU-Reg.
06464  58c1
06465  58c1
06466  58c1  ===> Test weitere Var. für CPU-Reg. <===
06467  58c1
06468  58c1  20 86 03    rreglp jsr chrgot        letztes Zeichen erneut nach ac
06469  58c4  f0 37              beq regrts        Trennzeichen, Ende
06470  58c6
06471  58c6  c9 2c              cmp #','
06472  58c8  f0 21              beq nxtreg        nächstes CPU-Reg.
06473  58ca
06474  58ca  ===> CPU-Reg. nach Var. <===
06475  58ca
06476  58ca  20 af 7a           jsr ptrget        folg. Variable suchen/anlegen
06477  58cd  85 4b              sta lstpnt
06478  58cf  84 4c              sty lstpnt+1
06479  58d1
06480  58d1  a5 0f              lda valtyp        Flag Variablentyp
06481  58d3  d0 29              bne jchker        Behandlung »?type msmatch error«
06482  58d5
06483  58d5  a4 0d              ldy count
06484  58d7  b9 06 00           lda areg,y        CPU-Reg.
06485  58da
06486  58da  c0 03              cpy #<pchi
06487  58dc  d0 02              bne rreg10        CPU-Reg. nach Var.
06488  58de
06489  58de  a5 05              lda sreg          Puffer 8502 CPU SR
06490  58e0
06491  58e0
06492  58e0  ===> CPU-Reg. nach Var. <===
06493  58e0
06494  58e0  a8          rreg10 tay
06495  58e1  a9 00              lda #>255
06496  58e3  20 3c 79           jsr givayv        16 Bit INT yr/ac nach GK in FAC
06497  58e6
06498  58e6  a5 10              lda intflg        Flag für Gleitkomma/Integer
06499  58e8  20 e3 53           jsr qintgr        GK/INT aus FAC nach Var.
06500  58eb
```

```
06501  58eb
06502  58eb
06502  58eb  ===> nächstes CPU-Reg. <===
06503  58eb
06504  58eb  e6 0d      nxtreg inc count        Reg. Zähler
06505  58ed
06506  58ed  a5 0d             lda count
06507  58ef  c9 04             cmp #4
06508  58f1  b0 0a             bcs regrts
06509  58f3
06510  58f3  ===> Test auf weitere Var. für CPU-Reg. <===
06511  58f3
06512  58f3  20 86 03          jsr chrgot      letztes Zeichen erneut nach ac
06513  58f6  f0 05             beq regrts      Trennzeichen
06514  58f8
06515  58f8  20 80 03          jsr chrget      nächstes Zeichen nach ac
06516  58fb  d0 c4             bne rreglp      (fast) immer
06517  58fd
06518  58fd  60         regrts rts
06519  58fe
06520  58fe
06521  58fe  ===> Behandlung »?type mismatch error« <===
06522  58fe
06523  58fe  4c e7 77   jchker jmp chkerr      Behandlung »?type msmatch error«
06524  5901
06525  5901
06526  5901  ===> Zuweisung MID$ <===
06527  5901
06528  5901  20 59 79   replc  jsr chkopn      Test auf folg. »(«, sonst Fehler
06529  5904
06530  5904   Syntax der Zuweisung mit MID$:
06531  5904
06532  5904   <let> mid$(x$,v <,l>) = y$
06533  5904
06534  5904   x$ = Zielstring
06535  5904   v  = Startspalte Ersatzstück in Zielstring
06536  5904   l  = Länge des Ersatzstücks
06537  5904   y$ = Ersatzstring
06538  5904
06539  5904   (Das dem Autor vorliegende Handbuch enthält keinen
06540  5904    Hinweis auf diese Anwendungsmöglichkeit von MID$)
06541  5904
06542  5904  20 af 7a          jsr ptrget      folg. Variable suchen/anlegen
06543  5907  85 4b             sta lstpnt
06544  5909  84 4c             sty lstpnt+1
06545  590b
06546  590b  20 dd 77          jsr chkstr      Test String VAR, sonst type mism.
06547  590e
06548  590e  ===> Startspalte Ersatzstück einlesen <===
06549  590e
06550  590e  20 09 88          jsr combyt      Kommatest, 8 Bit INT nach xr
06551  5911  ca                dex
06552  5912  86 78             stx midpt       Zeiger auf Start Ersatzstück
06553  5914
06554  5914  ===> Test Länge Ersatzstück angegeben <===
06555  5914
06556  5914  c9 29             cmp #')'
06557  5916  f0 04             beq repl10
```

```
06558  5918
06559  5918
06559  5918  ===> Länge Ersatzstück einlesen <===
06560  5918
06561  5918  20 09 88          jsr combyt       Länge Ersatz nach xr
06562  591b  2c                .byte _skip2
06563  591c
06564  591c  a2 ff       repl10 ldx #255         max. Stringlänge
06565  591e  86 77               stx midln        Länge Ersatzstück
06566  5920
06567  5920  20 56 79          jsr chkcls       Test auf folg. »)«, sonst Fehler
06568  5923
06569  5923  ===> Test auf folg. Gleichheitszeichen <===
06570  5923
06571  5923  a9 b2               lda #_eqult      GLEICH = Token
06572  5925  20 5e 79          jsr synchr       Test auf Zeichen in ac, sonst Fehler
06573  5928
06574  5928  ===> Zeiger auf Ersatzstring <===
06575  5928
06576  5928  20 ef 77          jsr frmevl       folgenden Ausdruck auswerten
06577  592b  20 dd 77          jsr chkstr       Test String VAR, sonst type mism.
06578  592e
06579  592e  a0 02               ldy #2
06580  5930  a9 4b       repl20 lda #<lstpnt
06581  5932  20 ab 03          jsr iram1        Desc. Zielstring
06582  5935  99 5d 00          sta str1,y       temp. Desc. Zielstring
06583  5938
06584  5938  20 e7 42          jsr indfmo       Desc. Ersatzstring
06585  593b  99 60 00          sta str2,y       temp. Desc. Ersatzstring
06586  593e
06587  593e  88                dey
06588  593f  10 ef             bpl repl20
06589  5941
06590  5941  38                sec
06591  5942  a5 61             lda str2+1       ADDRL Ersatzstring
06592  5944  e5 78             sbc midpt        Zeiger Start Ersatzstück
06593  5946  85 61             sta str2+1       ADDRL Ersatzstring
06594  5948
06595  5948  b0 02             bcs repl30
06596  594a  c6 62             dec str2+2       ADDRH Ersatzstring
06597  594c
06598  594c  a5 77       repl30 lda midln        Länge Ersatzstück
06599  594e  c5 60             cmp str2         Länge Ersatzstring
06600  5950  90 02             bcc repl40
06601  5952
06602  5952  a5 60             lda str2         Länge Ersatzstring
06603  5954
06604  5954  aa         repl40 tax              Länge Ersatzstück
06605  5955  f0 18             beq repend       Länge = 0, nicht ersetzen
06606  5957
06607  5957  18                clc
06608  5958  65 78             adc midpt        Zeiger Start Ersatzstück
06609  595a  b0 16             bcs jfcerr       Behandlung »?illegal quantity error«
06610  595c
06611  595c  c5 5d             cmp str1         Länge Zielstring
06612  595e  90 02             bcc repl50
06613  5960  d0 10             bne jfcerr       Behandlung »?illegal quantity error«
06614  5962
```

```
06615  5962  a4 78      repl50 ldy midpt        Zeiger Start Ersatzstück
06616  5964  a9 61      replop lda #<str2+1     Adresse Ersatzstring
06617  5966  20 ab 03          jsr iram1        Zeichen aus Ersatzstring
06618  5969  91 5e            sta (str1+1),y   nach Zielstring
06619  596b
06620  596b  c8                iny              Zeiger in Strings
06621  596c  ca                dex              Länge Ersatzstück
06622  596d  d0 f5              bne replop
06623  596f
06624  596f  4c 81 87   repend jmp frefac       Daten Arbeitsstring aus Ablage,
                                                Adresse index1, Länge ac
06625  5972
06626  5972
06627  5972  ===> Behandlung »?illegal quantity error« <===
06628  5972
06629  5972  4c 28 7d   jfcerr jmp fcerr        Behandlung »?illegal quantity error«
06630  5975
06631  5975
06632  5975  ===> BASIC-Statement AUTO ($dc) <===
06633  5975
06634  5975  20 f0 84   auto   jsr errind       Test Direktmodus, sonst Fehler
06635  5978
06636  5978  20 a0 50          jsr linget       Schrittweite nach linnum
06637  597b
06638  597b  a5 16            lda linnum
06639  597d  85 74            sta autinc       Schrittweite low für AUTO
06640  597f
06641  597f  a5 17            lda linnum+1
06642  5981  85 75            sta autinc+1     Schrittweite high für AUTO
06643  5983
06644  5983  4c 37 4d          jmp ready        Ausgabe »ready.« und ready-Status
06645  5986
06646  5986
06647  5986  ===> BASIC-Statement HELP ($ea) <===
06648  5986
06649  5986  ae 08 12   help   ldx errnum       Nummer Fehlertext
06650  5989  e8                inx
06651  598a  f0 1d            beq help01       $ff: Fehlernummer READY (ok)
06652  598c
06653  598c  ad 09 12          lda errlin       Low-Byte Zeilennummer für RESUME
06654  598f  ac 0a 12          ldy errlin+1     High-Byte Zeilennummer für RESUME
06655  5992
06656  5992  85 16            sta linnum
06657  5994  84 17            sty linnum+1
06658  5996
06659  5996  20 64 50          jsr fndlin       Adr. linnum/+1 nach lowtr/+1
06660  5999  90 0e            bcc help01       nicht gefunden
06661  599b
06662  599b  66 55            ror helper       HELP Flag setzen (CARRY=1)
06663  599d  20 98 55          jsr crdo         Ausgabe cr (+lf) auf akt. Kanal
06664  59a0
06665  59a0  a6 16            ldx linnum
06666  59a2  a5 17            lda linnum+1
06667  59a4  20 23 51          jsr p1line       LIST akt. Zeile
06668  59a7
06669  59a7  46 55            lsr helper       HELP Flag rücksetzen
06670  59a9
06671  59a9  4c 98 55   help01 jmp crdo         Ausgabe cr (+lf) auf akt. Kanal
```

```
06672  59ac
06673  59ac
06674  59ac
06674  59ac  ===> Test Fehlerspalte, RVS/BLINK ON <===
06675  59ac
06676  59ac  a6 62      helpsb ldx lowtr+1
06677  59ae
06678  59ae  98               tya             akt. Spalte in Fehlerzeile
06679  59af  18               clc
06680  59b0  65 61            adc lowtr
06681  59b2
06682  59b2  90 01            bcc lster2
06683  59b4  e8               inx
06684  59b5
06685  59b5  ec 0f 12   lster2 cpx errtxt+1    ADDRH txtptr bei Fehler
06686  59b8  d0 14            bne lstert
06687  59ba
06688  59ba  cd 0e 12         cmp errtxt       ADDRL txtptr bei Fehler
06689  59bd  90 0f            bcc lstert
06690  59bf  f0 0d            beq lstert
06691  59c1
06692  59c1  46 55            lsr helper       HELP Flag rücksetzen
06693  59c3
06694  59c3  ===> Attribut setzen <===
06695  59c3
06696  59c3  a9 12            lda #_rvson
06697  59c5
06698  59c5  24 d7            bit mode         BS-Modus 40/80-Zeichen
06699  59c7  10 02            bpl lster3       40 Zeichen BS
06700  59c9
06701  59c9  a9 02            lda #_blink
06702  59cb  4c 0c 56   lster3 jmp outdo        Ausgabe ac auf akt. Kanal
06703  59ce
06704  59ce
06705  59ce  60         lstert rts
06706  59cf
06707  59cf
06708  59cf  ===> BASIC-Statement GOSUB ($8d) <===
06709  59cf
06710  59cf  20 1d 5a   gosub  jsr gosusb       Textzeiger, Zeilennummer und
                                                gosub-Token auf Stapel
06711  59d2  20 86 03         jsr chrgot       letztes Zeichen erneut nach ac
06712  59d5  20 db 59         jsr goto         BASIC-Statement GOTO ($89)
06713  59d8  4c f6 4a         jmp newstt       Stopptaste abfragen, nächst.
                                                BASIC-Befehl/Zeile
06714  59db
06715  59db
06716  59db  ===> BASIC-Statement GOTO ($89) <===
06717  59db
06718  59db  20 a0 50   goto   jsr linget      Zeilennr. nach linnum
06719  59de
06720  59de  a5 0a            lda numflg       Flag Zeilennr. gelesen
06721  59e0  f0 38            beq numerr       Behandlung »?syntax error«
06722  59e2
06723  59e2  20 a5 52   goto_1 jsr remn        BASIC-Zeilenende suchen, Offset yr
06724  59e5
06725  59e5  ===> GOTO Zeile mit akt. Zeile vergleichen <===
06726  59e5
```

```
06727  59e5  38              sec
06728  59e6  a5 3b           lda curlin       Low-Byte akt. Zeilennummer
06729  59e8  e5 16           sbc linnum
06730  59ea
06731  59ea  a5 3c           lda curlin+1     High-Byte akt. Zeilennummer
06732  59ec  e5 17           sbc linnum+1
06733  59ee  b0 0b           bcs lukall       Zeilennr. < akt. suchen
06734  59f0
06735  59f0  ===> GOTO Zeilennr. > akt. Zeilennr. <===
06736  59f0
06737  59f0  98              tya              Zeiger hinter akt. Zeile
06738  59f1  38              sec
06739  59f2  65 3d           adc txtptr       ADDRL chrget-Zeiger
06740  59f4
06741  59f4  a6 3e           ldx txtptr+1     ADDRH chrget-Zeiger
06742  59f6  90 07           bcc luk4it       Zeilennr. > akt. suchen
06743  59f8
06744  59f8  e8              inx
06745  59f9  b0 04           bcs luk4it       immer
06746  59fb
06747  59fb
06748  59fb  ===> Zeilennr. < akt. suchen <===
06749  59fb
06750  59fb  a5 2d     lukall lda txttab       ADDRL Untergrenze akt. PGM
06751  59fd  a6 2e            ldx txttab+1     ADDRH Untergrenze akt. PGM
06752  59ff
06753  59ff
06754  59ff  ===> Zeilennr. > akt. suchen <===
06755  59ff
06756  59ff  20 68 50  luk4it jsr fndlnc       Adr. linnum nach lowtr, suchen ab
                                               ac/yr
06757  5a02  90 11           bcc userr        nicht gefunden
06758  5a04
06759  5a04  ===> Adresse GOTO Zeile nach txtptr <===
06760  5a04
06761  5a04  a5 61           lda lowtr
06762  5a06  e9 01           sbc #1
06763  5a08  85 3d           sta txtptr       ADDRL chrget-Zeiger
06764  5a0a
06765  5a0a  a5 62           lda lowtr+1
06766  5a0c  e9 00           sbc #0
06767  5a0e  85 3e           sta txtptr+1     ADDRH chrget-Zeiger
06768  5a10
06769  5a10  24 7f           bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
06770  5a12  10 6d           bpl setexc       PGM Modus setzen
06771  5a14
06772  5a14  60              rts
06773  5a15
06774  5a15
06775  5a15  ===> Behandlung »?undefined statement error« <===
06776  5a15
06777  5a15  a2 11     userr  ldx #_erus
06778  5a17  4c 3c 4d         jmp error        ind. Behandlung Fehlermeldung
06779  5a1a
06780  5a1a
06781  5a1a  ===> Behandlung »?syntax error« <===
06782  5a1a
06783  5a1a  4c 6c 79  numerr jmp snerr        Behandlung »?syntax error«
```

```
06784  5a1d
06785  5a1d
06786  5a1d
06786  5a1d  ===> Textzeiger, Zeilennummer und gosub-Token auf Stapel <===
06787  5a1d
06788  5a1d  a9 05      gosusb lda #_lngos      GOSUB Stapelbytes
06789  5a1f  20 fe 4f          jsr getstk      prüfen auf Platz im Stapel
06790  5a22
06791  5a22  ===> txtptr, curlin, GOSUB Token auf BASIC Stapel <===
06792  5a22
06793  5a22  a0 04              ldy #_lngos-1
06794  5a24  a5 3e              lda txtptr+1    ADDRH chrget-Zeiger
06795  5a26  91 7d              sta (tos),y
06796  5a28
06797  5a28  88                 dey
06798  5a29  a5 3d              lda txtptr      ADDRL chrget-Zeiger
06799  5a2b  91 7d              sta (tos),y
06800  5a2d
06801  5a2d  88                 dey
06802  5a2e  a5 3c              lda curlin+1    High-Byte akt. Zeilennummer
06803  5a30  91 7d              sta (tos),y
06804  5a32
06805  5a32  88                 dey
06806  5a33  a5 3b              lda curlin      Low-Byte akt. Zeilennummer
06807  5a35  91 7d              sta (tos),y
06808  5a37
06809  5a37  88                 dey
06810  5a38  a9 8d              lda #_gosut     GOSUB Token
06811  5a3a  91 7d              sta (tos),y
06812  5a3c
06813  5a3c  60                 rts
06814  5a3d
06815  5a3d
06816  5a3d  ===> BASIC-Statement GO ($cb) <===
06817  5a3d
06818  5a3d  20 80 03   gowoto jsr chrget      nächstes Zeichen nach ac
06819  5a40
06820  5a40  c9 a4              cmp #_tot       TO Token
06821  5a42  d0 06              bne go1         prüfen auf BASIC-Statement GO64
06822  5a44
06823  5a44  ===> BASIC-Statement GO TO ($cb $a4) <===
06824  5a44
06825  5a44  20 80 03           jsr chrget      nächstes Zeichen nach ac
06826  5a47  4c db 59           jmp goto        BASIC-Statement GOTO ($89)
06827  5a4a
06828  5a4a
06829  5a4a  ===> prüfen auf BASIC-Statement GO64 <===
06830  5a4a
06831  5a4a  20 f4 87   go1     jsr getbyt      8 Bit INT üb. frmnum nach xr
06832  5a4d  e0 40              cpx #64
06833  5a4f  f0 03              beq go64        BASIC-Statement GO64 ($cb ..)
06834  5a51
06835  5a51  ===> Behandlung »?syntax error« <===
06836  5a51
06837  5a51  4c 6c 79           jmp snerr       Behandlung »?syntax error«
06838  5a54
06839  5a54
```

```
06840  5a54  ===> BASIC-Statement GO64 ($cb ..) <===
06841  5a54
06842  5a54  20 e1 a7   go64   jsr rusure       Bearbeitung »are you sure ?«
06843  5a57  d0 06             bne go64rt       nicht »y«
06844  5a59
06845  5a59  ===> Umschaltung C64 - Modus <===
06846  5a59
06847  5a59  20 45 a8           jsr mapio        I/O-Chips einblenden
06848  5a5c  4c 4d ff           jmp kc64md       auf C64-Modus umschalten (Sprungv.)
06849  5a5f
06850  5a5f
06851  5a5f  60         go64rt rts
06852  5a60
06853  5a60
06854  5a60  ===> BASIC-Statement CONT ($9a) <===
06855  5a60
06856  5a60  d0 38      cont   bne contrt
06857  5a62
06858  5a62  24 7f             bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
06859  5a64  30 34             bmi contrt       CONT im PGM Modus (weiter)
06860  5a66
06861  5a66  ===> CONT im Direktmodus <===
06862  5a66
06863  5a66  a2 1a             ldx #_ercn       vorbesetzen »?can't continue«
06864  5a68
06865  5a68  ac 03 12          ldy oldtxt+1     ADDRH Zeiger letzter BASIC-Befehl
06866  5a6b  d0 03             bne contz3
06867  5a6d
06868  5a6d  ===> Behandlung »?can't continue error« <===
06869  5a6d
06870  5a6d  4c 3c 4d           jmp error        ind. Behandlung Fehlermeldung
06871  5a70
06872  5a70
06873  5a70  ad 02 12   contz3 lda oldtxt       ADDRL Zeiger letzter BASIC-Befehl
06874  5a73
06875  5a73  85 3d             sta txtptr       ADDRL chrget-Zeiger
06876  5a75  84 3e             sty txtptr+1     ADDRH chrget-Zeiger
06877  5a77
06878  5a77  ad 00 12          lda oldlin       Low-Byte vorige Zeilennummer (stop)
06879  5a7a  ac 01 12          ldy oldlin+1     High-Byte vorige Zeilennummer (stop)
06880  5a7d
06881  5a7d  85 3b             sta curlin       Low-Byte akt. Zeilennummer
06882  5a7f  84 3c             sty curlin+1     High-Byte akt. Zeilennummer
06883  5a81
06884  5a8106885  5a81  ===> PGM Modus setzen <===
06886  5a81
06887  5a81  a9 80      setexc lda #%10000000   PGM Modus
06888  5a83  85 7f             sta runmod       Modus: $80=PGM, $40=OVL, $00=DIR
06889  5a85
06890  5a85  0a                 asl a            = 0
06891  5a86  85 74             sta autinc       Schrittweite low für AUTO
06892  5a88  85 75             sta autinc+1     Schrittweite high für AUTO
06893  5a8a  8d 7f 12          sta intval       Flag für Bearbeitung COLLISION
06894  5a8d  85 f6             sta insflg       AUTO-INSERT-Flag
06895  5a8f
06896  5a8f  ===> BASIC IRQ Flags rücksetzen <===
06897  5a8f
06898  5a8f  a2 02             ldx #2
```

```
06899  5a91  9d 76 12   contlp sta tripfg,x     Sprite/Sprite    Coll.
06900  5a94  ca                 dex             Sprite/Hintergr. Coll.
06901  5a95  10 fa              bpl contlp      Light Pen
06902  5a97
06903  5a97  4c 90 ff           jmp kstmsg      Flag setzen Ausgabe/Unterdr.
                                                OPSYS-Meldung (Sprungv.)
06904  5a9a
06905  5a9a
06906  5a9a  60          contrt rts
06907  5a9b
06908  5a9b
06909  5a9b  ===> BASIC-Statement RUN ($8a) <===
06910  5a9b
06911  5a9b  f0 18       run    beq run10        RUN ohne Zeilennummer
06912  5a9d  90 1c              bcc run20        RUN mit Zeilennummer
06913  5a9f
06914  5a9f  ===> RUN mit Dateiname <===
06915  5a9f
06916  5a9f  a9 40              lda #%01000000
06917  5aa1  85 7f              sta runmod       Modus: $80=PGM, $40=OVL, $00=DIR
06918  5aa3
06919  5aa3  20 a7 a1           jsr dload        Disk-B.-Statem. DLOAD ($f0)
06920  5aa6
06921  5aa6
06922  5aa6  ===> RUN akt. PGM <===
06923  5aa6
06924  5aa6  20 81 5a   runprg jsr setexc       PGM Modus setzen
06925  5aa9  20 f3 51           jsr runc         BASIC-Zeiger initialisieren, CLR
06926  5aac  20 4f 4f           jsr lnkprg       alle Linkadressen neu berechnen
06927  5aaf  20 98 55           jsr crdo         Ausgabe cr (+lf) auf akt. Kanal
06928  5ab2  4c f6 4a           jmp newstt       Stopptaste abfragen, nächst.
                                                BASIC-Befehl/Zeile
06929  5ab5
06930  5ab5
06931  5ab5  ===> RUN ohne Zeilennummer <===
06932  5ab5
06933  5ab5  20 81 5a   run10  jsr setexc       PGM Modus setzen
06934  5ab8  4c f3 51           jmp runc         BASIC-Zeiger initialisieren, CLR
06935  5abb
06936  5abb
06937  5abb  ===> RUN mit Zeilennummer <===
06938  5abb
06939  5abb  20 fa 51   run20  jsr clearc       CLR durchführen
06940  5abe  20 86 03           jsr chrgot       letztes Zeichen erneut nach ac
06941  5ac1  20 db 59           jsr goto         BASIC-Statement GOTO ($89)
06942  5ac4  20 81 5a           jsr setexc       PGM Modus setzen
06943  5ac7  4c f6 4a           jmp newstt       Stopptaste abfragen, nächst.
                                                BASIC-Befehl/Zeile
06944  5aca
06945  5aca
06946  5aca  ===> BASIC-Statement RESTORE ($8c) <===
06947  5aca
06948  5aca  f0 15       restor beq resto1       RESTORE ohne Zeilennr.
06949  5acc
06950  5acc  ===> RESTORE mit Zeilennr. <===
06951  5acc
06952  5acc  20 12 88           jsr getwrd       16 Bit INT nach linnum
06953  5acf
```

```
06954  5acf  84 16            sty linnum
06955  5ad1  85 17            sta linnum+1
06956  5ad3
06957  5ad3  20 64 50         jsr fndlin       Adr. linnum/+1 nach lowtr/+1
06958  5ad6  b0 03            bcs rest1        lowtr/+1 nach DATA Zeiger
06959  5ad8
06960  5ad8
06960  5ad8  ===> Behandlung »?undefined statement error« <===
06961  5ad8
06962  5ad8  4c 15 5a         jmp userr
06963  5adb
06964  5adb
06965  5adb  ===> lowtr/+1 nach DATA Zeiger <===
06966  5adb
06967  5adb  a5 61      rest1  lda lowtr
06968  5add  a4 62             ldy lowtr+1
06969  5adf  b0 05             bcs resto2       immer
06970  5ae1
06971  5ae1
06972  5ae1  ===> READ-Zeiger auf BASIC-Anfang <===
06973  5ae1
06974  5ae1  38         resto1 sec
06975  5ae2  a5 2d             lda txttab       ADDRL Untergrenze akt. PGM
06976  5ae4  a4 2e             ldy txttab+1     ADDRH Untergrenze akt. PGM
06977  5ae6
06978  5ae6  e9 01      resto2 sbc #1
06979  5ae8  b0 01             bcs rest4
06980  5aea  88                dey
06981  5aeb
06982  5aeb  85 43      rest4  sta datptr       ADDRL Zeiger auf DATA (READ)
06983  5aed  84 44             sty datptr+1     ADDRH Zeiger auf DATA (READ)
06984  5aef
06985  5aef  60         rest5  rts
06986  5af0
06987  5af0
06988  5af0                    .end
06989  5af0                    .lib basic.renum
06990  5af0  put"@basic.renum"
```

```
06992  5af0
06993  5af0
06994  5af0  ===> Tabelle der Tokens für RENUMBER <===
06995  5af0
06996  5af0  89         testwd .byte _gotot     GOTO    Token
06997  5af1  8a                .byte _runt      RUN     Token
06998  5af2  8d                .byte _gosut     GOSUB   Token
06999  5af3  a7                .byte _thent     THEN    Token
07000  5af4  8c                .byte _restt     RESTORE Token
07001  5af5  d6                .byte _resut     RESUME  Token
07002  5af6  d7                .byte _trapt     TRAP    Token
07003  5af7  d5                .byte _elset     ELSE    Token
07004  5af8
07005  5af8
07006  5af8  ===> BASIC-Statement RENUMBER ($f8) <===
07007  5af8
07008  5af8  20 f0 84   renum  jsr errind       Test Direktmodus, sonst Fehler
07009  5afb
07010  5afb  ===> Schrittweite, alte und neue Startzeile vorbes. <===
07011  5afb
07012  5afb  a9 00             lda #>_deflt     Ersatzwert 10
07013  5afd  a2 0a             ldx #<_deflt
07014  5aff
07015  5aff  8e 70 11          stx rnstln       Startzeile neu
07016  5b02  8d 71 11          sta rnstln+1     mit 10 vorbesetzen
07017  5b05
07018  5b05  8e 72 11          stx rstep        Schrittweite
07019  5b08  8d 73 11          sta rstep+1      mit 10 vorbesetzen
07020  5b0b
07021  5b0b  85 5c             sta rostln       Startzeile alt
07022  5b0d  85 5d             sta rostln+1     mit 0 vorbesetzen
07023  5b0f
07024  5b0f  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
07025  5b12  f0 54             beq rpass1       RENUMBER ohne Parameter
07026  5b14
07027  5b14  ===> Startzeile neu (opt.) einlesen <===
07028  5b14
07029  5b14  20 a0 50          jsr linget       Zeilennr. nach linnum
07030  5b17
07031  5b17  a5 0a             lda numflg       Flag Zeilennr. gelesen
07032  5b19  f0 0a             beq rnum10       keine Eingabe
07033  5b1b
07034  5b1b  ===> Startzeile neu nach rnstln <===
07035  5b1b
07036  5b1b  a5 16             lda linnum
07037  5b1d  a6 17             ldx linnum+1
07038  5b1f
07039  5b1f  8d 70 11          sta rnstln       Low-Byte Startzeile neu
07040  5b22  8e 71 11          stx rnstln+1     High-Byte Startzeile neu
07041  5b25
07042  5b25
07043  5b25  ===> Schrittweite (opt.) einlesen <===
07044  5b25
07045  5b25  20 06 9e   rnum10 jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
07046  5b28  90 0e             bcc rnum20       keine Eingabe
07047  5b2a
07048  5b2a
```

```
07048  5b2a  ===> Schrittweite nach rstep <===
07049  5b2a
07050  5b2a  8c 72 11          sty rstep        Low-Byte Schrittweite
07051  5b2d  8d 73 11          sta rstep+1      High-Byte Schrittweite
07052  5b30
07053  5b30  ===> Test Schrittweite > 0 <===
07054  5b30
07055  5b30  0d 72 11          ora rstep        Low-Byte Schrittweite
07056  5b33  d0 03             bne rnum20       Startzeile alt (opt.) einlesen
07057  5b35
07058  5b35  ===> Behandlung »?illegal quantity error« <===
07059  5b35
07060  5b35  4c 28 7d   jfcer1 jmp fcerr        Behandlung »?illegal quantity error«
07061  5b38
07062  5b38
07063  5b38  ===> Startzeile alt (opt.) einlesen <===
07064  5b38
07065  5b38  20 06 9e   rnum20 jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
07066  5b3b  90 2b             bcc rpass1       keine Eingabe
07067  5b3d
07068  5b3d  ===> Startzeile alt nach rostln <===
07069  5b3d
07070  5b3d  84 5c             sty rostln       Low-Byte Startzeile alt
07071  5b3f  84 16             sty linnum
07072  5b41
07073  5b41  85 5d             sta rostln+1     High-Byte Startzeile alt
07074  5b43  85 17             sta linnum+1
07075  5b45
07076  5b45  ===> Adr. Startzeile alt nach lowtr <===
07077  5b45
07078  5b45  20 64 50          jsr fndlin       Adr. Startzeile alt nach lowtr
07079  5b48
07080  5b48  a5 61             lda lowtr
07081  5b4a  a6 62             ldx lowtr+1
07082  5b4c
07083  5b4c  85 5a             sta highds
07084  5b4e  86 5b             stx highds+1
07085  5b50
07086  5b50  ===> Adr. Startzeile neu nach lowtr <===
07087  5b50
07088  5b50  ad 70 11          lda rnstln       Low-Byte Startzeile neu
07089  5b53  ae 71 11          ldx rnstln+1     High-Byte Startzeile neu
07090  5b56
07091  5b56  85 16             sta linnum
07092  5b58  86 17             stx linnum+1
07093  5b5a
07094  5b5a  20 64 50          jsr fndlin       Adr. Startzeile neu nach lowtr
07095  5b5d
07096  5b5d  ===> Test Adr. Startz. neu >= Adr. Startz. alt <===
07097  5b5d
07098  5b5d  38                sec
07099  5b5e  a5 61             lda lowtr
07100  5b60  e5 5a             sbc highds
07101  5b62
07102  5b62  a5 62             lda lowtr+1
07103  5b64  e5 5b             sbc highds+1
07104  5b66  90 cd             bcc jfcer1       Adr. Startz. neu < Adr. Startz. alt
07105  5b68
```

```
07106  5b68
07107  5b68  ===> PASS 1: Test max. neue Nr. < 64000 <===
07108  5b68
07109  5b68  20 68 5d   rpass1 jsr nlnfac       Startz. neu nach FAC, txtptr PGM
                                                Anf.
07110  5b6b  20 9c 5d          jsr chargt       ADDRL Linkadresse
07111  5b6e
07112  5b6e  c8                iny
07113  5b6f  20 c9 03          jsr indtxt       ADDRH Linkadresse
07114  5b72  f0 3a             beq rpass2       PGM Ende, Pass 2
07115  5b74
07116  5b74
07117  5b74  ===> Startzeile alt in PGM suchen <===
07118  5b74
07119  5b74  c8         pas1_1 iny
07120  5b75  20 c9 03          jsr indtxt       Low-Byte Zeilennr. nach ac
07121  5b78
07122  5b78  38                sec
07123  5b79  e5 5c             sbc rostln       Low-Byte Startzeile alt
07124  5b7b
07125  5b7b  c8                iny
07126  5b7c  20 c9 03          jsr indtxt       High-Byte Zeilennr. nach ac
07127  5b7f  e5 5d             sbc rostln+1     High-Byte Startzeile alt
07128  5b81  b0 07             bcs pas1_2       akt. Zeilennr. >= Startzeile alt
07129  5b83
07130  5b83  20 9d 5b          jsr setnxt       txtptr auf nächsten Zeilenkopf
07131  5b86  d0 ec             bne pas1_1       Startzeile alt in PGM suchen
07132  5b88  f0 24             beq rpass2       immer
07133  5b8a
07134  5b8a
07135  5b8a  ===> Test auf neue Zeilennr. < 64000 <===
07136  5b8a
07137  5b8a  20 9d 5b   pas1_2 jsr setnxt       txtptr auf nächsten Zeilenkopf
07138  5b8d  f0 1f             beq rpass2       PGM Ende, Pass 2
07139  5b8f
07140  5b8f  20 89 5d          jsr newnum       FAC plus Schrittweite
07141  5b92  b0 04             bcs errnlg       Behandlung »?line number too large
                                                error«
07142  5b94
07143  5b94  c9 f9             cmp #>63744
07144  5b96
07145  5b96  ***************************  BUG  ****************************
07146  5b96  *  müßte eigentlich  >64000 heißen, dieser BUG erzeugt die    *
07147  5b96  *  Fehlermeldung »?line number too large« für Nummern >63743  *
07148  5b96  ************************************************************
07149  5b96
07150  5b96  90 f2             bcc pas1_2       Test auf neue Zeilennr. < 64000
07151  5b98
07152  5b98
07153  5b98  ===> Behandlung »?line number too large error« <===
07154  5b98
07155  5b98  a2 26      errnlg ldx #_erntl
07156  5b9a  4c 3c 4d          jmp error        ind. Behandlung Fehlermeldung
07157  5b9d
07158  5b9d
07159  5b9d  ===> txtptr auf nächsten Zeilenkopf <===
07160  5b9d
07161  5b9d  a0 00      setnxt ldy #0
```

```
07162  5b9f  20 c9 03          jsr indtxt       ADDRL Linkadresse
07163  5ba2  aa                tax
07164  5ba3
07165  5ba3  c8                iny
07166  5ba4  20 c9 03          jsr indtxt       ADDRH Linkadresse
07167  5ba7  f0 04             beq setend       PGM Ende
07168  5ba9
07169  5ba9  86 3d             stx txtptr       ADDRL chrget-Zeiger
07170  5bab  85 3e             sta txtptr+1     ADDRH chrget-Zeiger
07171  5bad
07172  5bad  60          setend rts
07173  5bae
07174  5bae
07175  5bae  ===> PASS 2: Test / Zeilennr. ersetzen <===
07176  5bae
07177  5bae  a9 01       rpass2 lda #1           Flag: Probelauf
07178  5bb0  85 77              sta rnumfg
07179  5bb2
07180  5bb2  ad 10 12           lda txttop       ADDRL Obergrenze akt. PGM
07181  5bb5  ae 11 12           ldx txttop+1     ADDRH Obergrenze akt. PGM
07182  5bb8
07183  5bb8  85 3f              sta fndpnt       txttop
07184  5bba  86 40              stx fndpnt+1     txttop+1
07185  5bbc
07186  5bbc  20 fb 5b           jsr imbdln       neue Zeilennr. ber., RAM Test
07187  5bbf
07188  5bbf  c6 77              dec rnumfg       Flag: Zeilennr. ersetzen
07189  5bc1
07190  5bc1  20 fb 5b           jsr imbdln       neue Zeilennr. ber., ersetzen
07191  5bc4
07192  5bc4  20 99 5d   pas2_1  jsr chgtx2       Linkadresse überlesen
07193  5bc7  f0 2f              beq rnuxit       PGM Ende, Ende RENUMBER
07194  5bc9
07195  5bc9  20 9c 5d           jsr chargt       nächst. Zeichen üb. txtptr nach ac
07196  5bcc  85 16              sta linnum       Low-Byte Zeilennr.
07197  5bce
07198  5bce  c8                 iny
07199  5bcf  20 c9 03           jsr indtxt       High-Byte Zeilennr.
07200  5bd2
07201  5bd2  ===> Test Zeilennr. in RENUMBER Bereich <===
07202  5bd2
07203  5bd2  38                 sec
07204  5bd3  e5 5d              sbc rostln+1     High-Byte Startzeile alt
07205  5bd5  90 19              bcc pas2_3       nicht in RENUMBER Bereich
07206  5bd7  d0 06              bne pas2_2       neue Zeilennr. in Zeilenkopf
07207  5bd9
07208  5bd9  a5 16              lda linnum
07209  5bdb  e5 5c              sbc rostln       Low-Byte Startzeile alt
07210  5bdd  90 11              bcc pas2_3       nicht in RENUMBER Bereich
07211  5bdf
07212  5bdf
07213  5bdf  ===> neue Zeilennr. in Zeilenkopf <===
07214  5bdf
07215  5bdf  a5 64       pas2_2 lda facho        High-Byte neue Zeilennr.
07216  5be1  91 3d              sta (txtptr),y   in Zeilenkopf
07217  5be3
07218  5be3  88                 dey
07219  5be4  a5 65              lda facmoh       Low-Byte neue Zeilennr.
```

```
07220  5be6  91 3d            sta (txtptr),y   in Zeilenkopf
07221  5be8
07222  5be8  20 9c 5d          jsr chargt       nächst. Zeichen üb. txtptr nach ac
07223  5beb  20 80 5d          jsr lininc       neue Zeilennr. berechn., txtptr
                                                Zeilenende
07224  5bee  f0 d4             beq pas2_1       immer
07225  5bf0
07226  5bf0
07227  5bf0  ===> txtptr auf Zeilenende <===
07228  5bf0
07229  5bf0  20 9c 5d   pas2_3 jsr chargt       nächst. Zeichen üb. txtptr nach ac
07230  5bf3  20 83 5d          jsr scnthr       txtptr auf Zeilenende
07231  5bf6  f0 cc             beq pas2_1       immer
07232  5bf8
07233  5bf8
07234  5bf8  4c e5 5e   rnuxit jmp notdel       Ende RENUMBER
07235  5bfb
07236  5bfb
07237  5bfb  ===> neue Zeilennr. ber., RAM Test/ersetzen <===
07238  5bfb
07239  5bfb  20 54 52   imbdln jsr stxtpt       txtptr auf PGM Anf.
07240  5bfe  20 99 5d   nxtlin jsr chgtx2       chargt, chargt
07241  5c01  d0 03             bne imbd10       nicht PGM Ende
07242  5c03
07243  5c03  ===> PGM Ende <===
07244  5c03
07245  5c03  4c 68 5d          jmp nlnfac       Startz. neu nach FAC, txtptr PGM
                                                Anf.
07246  5c06
07247  5c06
07248  5c06  ===> Zeilennr. nach lstpnt <===
07249  5c06
07250  5c06  20 9c 5d   imbd10 jsr chargt       nächst. Zeichen üb. txtptr nach ac
07251  5c09  85 4b             sta lstpnt       Low-Byte Zeilennr.
07252  5c0b
07253  5c0b  20 9c 5d          jsr chargt       nächst. Zeichen üb. txtptr nach ac
07254  5c0e  85 4c             sta lstpnt+1     High-Byte Zeilennr.
07255  5c10
07256  5c10
07257  5c10  ===> Zeile nach Zeilennr. durchsuchen <===
07258  5c10
07259  5c10  20 9c 5d   nxtchr jsr chargt       nächst. Zeichen üb. txtptr nach ac
07260  5c13
07261  5c13
07262  5c13  ===> Test auf erstes Anführungszeichen <===
07263  5c13
07264  5c13  c9 22      chkqut cmp #'"'
07265  5c15  d0 0b             bne notqut
07266  5c17
07267  5c17
07268  5c17  ===> zweites Anführungszeichen suchen <===
07269  5c17
07270  5c17  20 9c 5d   nxtqut jsr chargt       nächst. Zeichen üb. txtptr nach ac
07271  5c1a  f0 e2             beq nxtlin       Zeilenende
07272  5c1c
07273  5c1c  c9 22             cmp #'"'
07274  5c1e  d0 f7             bne nxtqut       zweites Anführungszeichen suchen
07275  5c20  f0 ee             beq nxtchr       immer
```

```
07276  5c22
07277  5c22
07278  5c22  aa          notqut tax              Flags setzen
07279  5c23  f0 d9              beq nxtlin        Zeilenende
07280  5c25  10 e9              bpl nxtchr        kein Token
07281  5c27
07282  5c27  ===> Token, Test auf Token mit Zeilennummer <===
07283  5c27
07284  5c27  a2 08              ldx #renum-testwd
07285  5c29  dd ef 5a    ctoklp cmp testwd-1,x    Tabelle der Tokens für RENUMBER
07286  5c2c  f0 28              beq ilin10        Token mit Zeilennummer
07287  5c2e
07288  5c2e  ca                 dex
07289  5c2f  d0 f8              bne ctoklp
07290  5c31
07291  5c31  ===> Test auf GO TO ($cb $a4) <===
07292  5c31
07293  5c31  c9 cb              cmp #_got         GO Token
07294  5c33  d0 0b              bne chkesc        Test auf COLLISION Token ($fe $17)
07295  5c35
07296  5c35
07297  5c35  ===> Test auf TO Token <===
07298  5c35
07299  5c35  20 80 03    hop_1  jsr chrget       nächstes Zeichen nach ac
07300  5c38  f0 c4              beq nxtlin
07301  5c3a
07302  5c3a  c9 a4              cmp #_tot         TO Token
07303  5c3c  f0 18              beq ilin10        Token mit Zeilennummer
07304  5c3e  d0 d0              bne nxtchr        nicht Go TO
07305  5c40
07306  5c40
07307  5c40  ===> Test auf COLLISION Token ($fe $17) <===
07308  5c40
07309  5c40  c9 fe       chkesc cmp #_ecmdt       ESC COMMAND Token
07310  5c42  d0 cc              bne nxtchr        Zeile nach Zeilennr. durchsuchen
07311  5c44
07312  5c44  20 80 03           jsr chrget        nächstes Zeichen nach ac
07313  5c47  f0 ec              beq hop_1         Test auf TO Token
07314  5c49
07315  5c49  c9 17              cmp #_colit       COLLISION Token
07316  5c4b  d0 c3              bne nxtchr        Zeile nach Zeilennr. durchsuchen
07317  5c4d
07318  5c4d
07319  5c4d  ===> COLLISION gefunden, Kommas überlesen <===
07320  5c4d
07321  5c4d  20 80 03    fndesc jsr chrget       nächstes Zeichen nach ac
07322  5c50  f0 e3              beq hop_1         COLLISION ohne Zeilennr.
07323  5c52
07324  5c52  c9 2c              cmp #','
07325  5c54  d0 f7              bne fndesc        COLLISION gefunden, Kommas überlesen
07326  5c56
07327  5c56
07328  5c56  ===> Token mit Zeilennummer <===
07329  5c56
07330  5c56  a5 3d       ilin10 lda txtptr       ADDRL chrget-Zeiger
07331  5c58  8d 00 12           sta oldlin        txtptr
07332  5c5b
07333  5c5b  a5 3e              lda txtptr+1      ADDRH chrget-Zeiger
```

```
07334  5c5d  8d 01 12          sta oldlin+1     txtptr+1
07335  5c60
07336  5c60
07337  5c60  ===> Test auf Zeilennr. hinter Token <===
07338  5c60  20 80 03          jsr chrget       nächstes Zeichen nach ac
07339  5c63  b0 ae             bcs chkqut       keine Dez.Ziffer
07340  5c65
07341  5c65  ===>
07342  5c65
07343  5c65  20 a0 50          jsr linget       Zeilennr. nach linnum
07344  5c68  20 19 5d          jsr frmlin       Test/Ziel Zeilennr. neu berechnen
07345  5c6b
07346  5c6b  ad 00 12          lda oldlin       txtptr
07347  5c6e  85 3d             sta txtptr       ADDRL chrget-Zeiger
07348  5c70
07349  5c70  ad 01 12          lda oldlin+1     txtptr+1
07350  5c73  85 3e             sta txtptr+1     ADDRH chrget-Zeiger
07351  5c75
07352  5c75  20 80 03          jsr chrget       nächstes Zeichen nach ac
07353  5c78
07354  5c78  a5 3d             lda txtptr       ADDRL chrget-Zeiger
07355  5c7a  d0 02             bne ilin15
07356  5c7c  c6 3e             dec txtptr+1     ADDRH chrget-Zeiger
07357  5c7e  c6 3d      ilin15 dec txtptr       ADDRL chrget-Zeiger
07358  5c80
07359  5c80  a2 ff             ldx #255         Zeiger in Zeilennr. String
07360  5c82
07361  5c82  a5 77             lda rnumfg       Flag für Probelauf
07362  5c84
07363  5c84   0 = Zeilennr. ersetzen
07364  5c84   1 = Test genüg. RAM
07365  5c84
07366  5c84  f0 41             beq p3code       String Zeilennr. in PGM eintragen
07367  5c86
07368  5c86  ===> Probelauf, Test genüg. RAM für RENUMBER <===
07369  5c86
07370  5c86  20 8f 5c          jsr p2code       Test genügend RAM für neue Nr.
07371  5c89
07372  5c89
07373  5c89  ===> Test weitere Zeilennr. <===
07374  5c89
07375  5c89  c9 2c      ilin20 cmp #','         Komma vor weiterer Zeilennr.
07376  5c8b  f0 c9             beq ilin10       Token mit Zeilennummer
07377  5c8d  d0 84             bne chkqut       immer
07378  5c8f
07379  5c8f
07380  5c8f  ===> Test genügend RAM für neue Nr. <===
07381  5c8f
07382  5c8f  e8         p2code inx              Zeiger in Zeilennr. String
07383  5c90
07384  5c90  bd 01 01          lda fbuffr+1,x   String neue Zeilennr.
07385  5c93  f0 1f             beq p2cd30       Zeilennr. neu kürzer, txttop nachf.
07386  5c95
07387  5c95  20 80 03          jsr chrget       Zeichen alte Zeilennr.
07388  5c98  90 f5             bcc p2code       Dez.Ziffer
07389  5c9a
07390  5c9a
```

```
07391  5c9a
07391  5c9a  ===> Zeilennr. neu länger, txttop nachf. <===
07392  5c9a
07393  5c9a  e6 3f      p2cd10 inc fndpnt       txttop
07394  5c9c  d0 02             bne p2cd20       Test txttop < mxmem0
07395  5c9e  e6 40             inc fndpnt+1     txttop+1
07396  5ca0
07397  5ca0
07398  5ca0  ===> Test txttop < mxmem0 <===
07399  5ca0
07400  5ca0  38         p2cd20 sec
07401  5ca1  a5 3f             lda fndpnt       txttop
07402  5ca3  ed 12 12          sbc mxmem0       ADDRL Obergr. PGM RAM
07403  5ca6
07404  5ca6  a5 40             lda fndpnt+1     txttop+1
07405  5ca8  ed 13 12          sbc mxmem0+1     ADDRH Obergr. PGM RAM
07406  5cab  b0 17             bcs omerr1       Behandlung »?out of memory error«
07407  5cad
07408  5cad  e8                inx              Zeiger in String neue Zeilennr.
07409  5cae  bd 01 01          lda fbuffr+1,x
07410  5cb1  d0 e7             bne p2cd10       txttop nachführen
07411  5cb3
07412  5cb3  60                rts              _eom gefunden
07413  5cb4
07414  5cb4
07415  5cb4  ===> Zeilennr. neu kürzer, txttop nachf. <===
07416  5cb4
07417  5cb4  20 80 03   p2cd30 jsr chrget       nächstes Zeichen nach ac
07418  5cb7  b0 0a             bcs p2cd50       keine Dez.Ziffer
07419  5cb9
07420  5cb9  a5 3f             lda fndpnt       txttop
07421  5cbb  d0 02             bne p2cd40
07422  5cbd  c6 40             dec fndpnt+1     txttop+1
07423  5cbf
07424  5cbf  c6 3f      p2cd40 dec fndpnt       txttop
07425  5cc1  90 f1             bcc p2cd30       immer
07426  5cc3
07427  5cc3  60         p2cd50 rts
07428  5cc4
07429  5cc4
07430  5cc4  ===> Behandlung »?out of memory error« <===
07431  5cc4
07432  5cc4  4c 3a 4d   omerr1 jmp omerr        Behandlung »?out of memory error«
07433  5cc7
07434  5cc7
07435  5cc7  ===> String Zeilennr. in PGM eintragen <===
07436  5cc7
07437  5cc7  e8         p3code inx              Zeiger in String neue Zeilennr.
07438  5cc8
07439  5cc8  bd 01 01   p3cd10 lda fbuffr+1,x
07440  5ccb  f0 2c             beq p3cd30       neue Zeilennr. eingetr., Test
                                                versch.
07441  5ccd
07442  5ccd  48                pha              Ziffer Zeilennr.
07443  5cce
07444  5cce  20 9c 5d          jsr chargt       nächst. Zeichen üb. txtptr nach ac
07445  5cd1  c9 3a             cmp #$3a
07446  5cd3  b0 0c             bcs p3cd15       ASCII > »9«, verschieben
```

```
07447  5cd5
07448  5cd5  c9 20           cmp #_spc
07449  5cd7  f0 08           beq p3cd15       Space, verschieben
07450  5cd9
07451  5cd9  ===> Test ac numerisch <===
07452  5cd9
07453  5cd9  38              sec
07454  5cda  e9 30           sbc #$30
07455  5cdc
07456  5cdc  38              sec
07457  5cdd  e9 d0           sbc #256-$30
07458  5cdf  90 10           bcc p3cd20       numerisch
07459  5ce1
07460  5ce1
07461  5ce1  ===> PGM eine Adr. nach oben schieben <===
07462  5ce1
07463  5ce1  20 a7 5d   p3cd15 jsr movini      Verschiebeparam. laden
07464  5ce4  e6 6d             inc movspt      Verschiebezeiger Schreiben
07465  5ce6  20 df 5d          jsr moveup      PGM nach oben verschieben
07466  5ce9
07467  5ce9  ===> txttop nachführen <===
07468  5ce9
07469  5ce9  ee 10 12          inc txttop      ADDRL Obergrenze akt. PGM
07470  5cec  d0 03             bne p3cd20      Ziffer neue Zeilennr. eintragen
07471  5cee  ee 11 12          inc txttop+1    ADDRH Obergrenze akt. PGM
07472  5cf1
07473  5cf1
07474  5cf1  ===> Ziffer neue Zeilennr. eintragen <===
07475  5cf1
07476  5cf1  68         p3cd20 pla             Ziffer neue Zeilennr.
07477  5cf2
07478  5cf2  a0 00             ldy #0
07479  5cf4  91 3d             sta (txtptr),y  in PGM eintragen
07480  5cf6
07481  5cf6  e8                inx             Zeiger in String neue Zeilennr.
07482  5cf7  d0 cf             bne p3cd10      immer
07483  5cf9
07484  5cf9
07485  5cf9  ===> neue Zeilennr. eingetr., Test versch. <===
07486  5cf9
07487  5cf9  20 80 03   p3cd30 jsr chrget      nächstes Zeichen nach ac
07488  5cfc  b0 8b             bcs ilin20      keine Dez.Ziffer, fertig
07489  5cfe
07490  5cfe
07491  5cfe  ===> PGM bis Ende alte Zeilennr. nach unt. schieben <===
07492  5cfe
07493  5cfe  20 a7 5d   p3cd40 jsr movini      Verschiebeparam. laden
07494  5d01  c6 6d             dec movspt      Verschiebezeiger Schreiben
07495  5d03  20 c6 5d          jsr movedn      PGM nach unten verschieben
07496  5d06
07497  5d06  ===> txttop nachführen <===
07498  5d06
07499  5d06  ad 10 12          lda txttop      ADDRL Obergrenze akt. PGM
07500  5d09  d0 03             bne p3cd50
07501  5d0b  ce 11 12          dec txttop+1    ADDRH Obergrenze akt. PGM
07502  5d0e  ce 10 12   p3cd50 dec txttop      ADDRL Obergrenze akt. PGM
07503  5d11
07504  5d11  20 86 03          jsr chrgot      letztes Zeichen erneut nach ac
```

```
07505  5d14  90 e8            bcc p3cd40       Dez.Ziffer, weiter verschieben
07506  5d16
07507  5d16  4c 89 5c         jmp ilin20       Test weitere Zeilennr.
07508  5d19
07509  5d19
07510  5d19  ===> Test/Ziel Zeilennr. neu berechnen <===
07511  5d19
07512  5d19  20 68 5d   frmlin jsr nlnfac      Startz. neu nach FAC, txtptr PGM
                                               Anf.
07513  5d1c  20 99 5d   findit jsr chgtx2      chargt, chargt
07514  5d1f  d0 0d            bne find10
07515  5d21
07516  5d21  ===> Behandlung »?unresolved reference error« <===
07517  5d21
07518  5d21  a2 27            ldx #_erref
07519  5d23
07520  5d23  a5 4b            lda lstpnt       Low-Byte Fehlerzeile
07521  5d25  85 3b            sta curlin       Low-Byte akt. Zeilennummer
07522  5d27
07523  5d27  a5 4c            lda lstpnt+1     High-Byte Fehlerzeile
07524  5d29  85 3c            sta curlin+1     High-Byte akt. Zeilennummer
07525  5d2b
07526  5d2b  4c 3c 4d         jmp error        ind. Behandlung Fehlermeldung
07527  5d2e
07528  5d2e
07529  5d2e  20 9c 5d   find10 jsr chargt      nächst. Zeichen üb. txtptr nach ac
07530  5d31  85 5a            sta highds       Low-Byte Zeilennr.
07531  5d33
07532  5d33  c5 16            cmp linnum       Low-Byte Ziel Zeilennr.
07533  5d35  d0 27            bne find40       nicht Ziel Zeile, weitersuchen
07534  5d37
07535  5d37  20 9c 5d         jsr chargt       nächst. Zeichen üb. txtptr nach ac
07536  5d3a  85 5b            sta highds+1     High-Byte Zeilennr.
07537  5d3c
07538  5d3c  c5 17            cmp linnum+1     High-Byte Ziel Zeilennr.
07539  5d3e  d0 23            bne find50
07540  5d40
07541  5d40  ===> Ziel Zeile gefunden, Test in RENUMBER Bereich <===
07542  5d40
07543  5d40  38               sec
07544  5d41  e5 5d            sbc rostln+1     High-Byte Startzeile alt
07545  5d43  90 08            bcc find20       alte Ziel Zeilennr. nach FAC
07546  5d45  d0 0e            bne find30       Ziel Zeilennr. nach String
07547  5d47
07548  5d47  a5 16            lda linnum
07549  5d49  e5 5c            sbc rostln       Low-Byte Startzeile alt
07550  5d4b  b0 08            bcs find30       Ziel Zeilennr. in RENUMBER Bereich
07551  5d4d
07552  5d4d
07553  5d4d  ===> alte Ziel Zeilennr. nach FAC <===
07554  5d4d
07555  5d4d  a5 16      find20 lda linnum       Low-Byte alte Ziel Zeilennr.
07556  5d4f  85 65            sta facmoh       FAC: M3
07557  5d51
07558  5d51  a5 17            lda linnum+1     High-Byte alte Ziel Zeilennr.
07559  5d53  85 64            sta facho        FAC: M4
07560  5d55
07561  5d55
```

```
07562  5d55
07562  5d55  ===> Ziel Zeilennr. nach String <===
07563  5d55
07564  5d55  a2 90      find30 ldx #_expcn+15
07565  5d57  38               sec
07566  5d58  20 75 8c          jsr floatc       EX nach FAC, Nullen auffüllen,
                                                normal.
07567  5d5b  4c 42 8e          jmp fout         String Ziel Zeilennr. nach fbuffr
07568  5d5e
07569  5d5e
07570  5d5e  ===> nicht Ziel Zeile, weitersuchen <===
07571  5d5e
07572  5d5e  20 9c 5d   find40 jsr chargt       nächst. Zeichen üb. txtptr nach ac
07573  5d61  85 5b             sta highds+1     High-Byte Zeilennr.
07574  5d63
07575  5d63  20 75 5d   find50 jsr linadd       Test / newnum, txtptr Zeilenende
07576  5d66  f0 b4             beq findit       immer
07577  5d68
07578  5d68
07579  5d68  ===> Startz. neu nach FAC, txtptr PGM Anf. <===
07580  5d68
07581  5d68  ad 70 11   nlnfac lda rnstln       Low-Byte Startzeile neu
07582  5d6b  85 65             sta facmoh       FAC: M3
07583  5d6d
07584  5d6d  ad 71 11          lda rnstln+1     High-Byte Startzeile neu
07585  5d70  85 64             sta facho        FAC: M4
07586  5d72
07587  5d72  4c 54 52          jmp stxtpt       txtptr auf PGM Anf.
07588  5d75
07589  5d75
07590  5d75  ===> Test / newnum, txtptr Zeilenende <===
07591  5d75
07592  5d75  a5 5a      linadd lda highds       ADDRL Start alt
07593  5d77  38               sec
07594  5d78  e5 5c             sbc rostln       Low-Byte Startzeile alt
07595  5d7a
07596  5d7a  a5 5b             lda highds+1     ADDRH Start alt
07597  5d7c  e5 5d             sbc rostln+1     High-Byte Startzeile alt
07598  5d7e  90 03             bcc scnthr       txtptr auf Zeilenende
07599  5d80
07600  5d80
07601  5d80  ===> neue Zeilennr. berechn., txtptr Zeilenende <===
07602  5d80
07603  5d80  20 89 5d   lininc jsr newnum       FAC plus Schrittweite
07604  5d83
07605  5d83
07606  5d83  ===> txtptr auf Zeilenende <===
07607  5d83
07608  5d83  20 9c 5d   scnthr jsr chargt       nächst. Zeichen üb. txtptr nach ac
07609  5d86  d0 fb             bne scnthr       txtptr auf Zeilenende
07610  5d88
07611  5d88  60               rts
07612  5d89
07613  5d89
07614  5d89  ===> FAC plus Schrittweite <===
07615  5d89
07616  5d89  a5 65      newnum lda facmoh       FAC: M3
07617  5d8b  18               clc
```

```
07618  5d8c  6d 72 11          adc rstep       Low-Byte Schrittweite
07619  5d8f  85 65             sta facmoh      FAC: M3
07620  5d91
07621  5d91  a5 64             lda facho       FAC: M4
07622  5d93  6d 73 11          adc rstep+1     High-Byte Schrittweite
07623  5d96  85 64             sta facho       FAC: M4
07624  5d98
07625  5d98  60                rts
07626  5d99
07627  5d99
07628  5d99  ===> chargt, chargt <===
07629  5d99
07630  5d99  20 9c 5d   chgtx2 jsr chargt      nächst. Zeichen üb. txtptr nach ac
07631  5d9c
07632  5d9c
07633  5d9c  ===> nächst. Zeichen üb. txtptr nach ac <===
07634  5d9c
07635  5d9c  a0 00      chargt ldy #0
07636  5d9e  e6 3d             inc txtptr      ADDRL chrget-Zeiger
07637  5da0  d0 02             bne chrgrt
07638  5da2  e6 3e             inc txtptr+1    ADDRH chrget-Zeiger
07639  5da4  4c c9 03   chrgrt jmp indtxt      LDA (txtptr),y aus PGM-RAM
07640  5da7
07641  5da7
07642  5da7  ===> Verschiebeparam. laden <===
07643  5da7
07644  5da7  a5 3d      movini lda txtptr       ADDRL chrget-Zeiger
07645  5da9  85 24             sta index1      ADDRL Verschiebezeiger
07646  5dab
07647  5dab  a5 3e             lda txtptr+1    ADDRH chrget-Zeiger
07648  5dad  85 25             sta index1+1    ADDRH Verschiebezeiger
07649  5daf
07650  5daf  ad 10 12          lda txttop      ADDRL Obergrenze akt. PGM
07651  5db2  85 26             sta index2      ADDRL Verschiebe Ende
07652  5db4
07653  5db4  ad 11 12          lda txttop+1    ADDRH Obergrenze akt. PGM
07654  5db7  85 27             sta index2+1    ADDRH Verschiebe Ende
07655  5db9
07656  5db9  a0 00             ldy #0
07657  5dbb  84 0d             sty movlpt      Verschiebezeiger Lesen
07658  5dbd  84 6d             sty movspt      Verschiebezeiger Schreiben
07659  5dbf
07660  5dbf  60                rts
07661  5dc0
07662  5dc0
07663  5dc0  e6 24      mvdnlp inc index1
07664  5dc2  d0 02             bne movedn      PGM nach unten verschieben
07665  5dc4  e6 25             inc index1+1
07666  5dc6
07667  5dc6
07668  5dc6  ===> PGM nach unten verschieben <===
07669  5dc6
07670  5dc6  a4 0d      movedn ldy movlpt       Verschiebezeiger Lesen
07671  5dc8  c8                iny
07672  5dc9  20 05 43          jsr ind1_0      LDA (index1),y aus PGM-RAM
07673  5dcc
07674  5dcc  a4 6d             ldy movspt      Verschiebezeiger Schreiben
07675  5dce  c8                iny
```

```
07676  5dcf  91 24            sta (index1),y
07677  5dd1
07678  5dd1  20 ee 5d         jsr compar       Test Verschiebe Ende erreicht
07679  5dd4  d0 ea            bne mvdnlp
07680  5dd6
07681  5dd6  60               rts
07682  5dd7
07683  5dd7
07684  5dd7  a5 26      mvuplp lda index2
07685  5dd9  d0 02            bne mvup10
07686  5ddb  c6 27            dec index2+1
07687  5ddd  c6 26      mvup10 dec index2
07688  5ddf
07689  5ddf
07690  5ddf  ===> PGM nach oben verschieben <===
07691  5ddf
07692  5ddf  a4 0d      moveup ldy movlpt       Verschiebezeiger Lesen
07693  5de1  20 c0 03         jsr ind2_0       LDA (index2),y aus PGM-RAM
07694  5de4
07695  5de4  a4 6d            ldy movspt       Verschiebezeiger Schreiben
07696  5de6  91 26            sta (index2),y
07697  5de8
07698  5de8  20 ee 5d         jsr compar       Test Verschiebe Ende erreicht
07699  5deb  d0 ea            bne mvuplp
07700  5ded
07701  5ded  60               rts
07702  5dee
07703  5dee
07704  5dee  ===> Test Verschiebe Ende erreicht <===
07705  5dee
07706  5dee  a5 24      compar lda index1
07707  5df0
07708  5df0  c5 26            cmp index2
07709  5df2  d0 04            bne cprts
07710  5df4
07711  5df4  a5 25            lda index1+1
07712  5df6  c5 27            cmp index2+1
07713  5df8  60         cprts  rts
07714  5df9
07715  5df9
07716  5df9                   .end
07717  5df9                   .lib basic.cmd_4
07718  5df9  put"@basic.cmd_4"
```

```
07720  5df9
07721  5df9
07722  5df9  ===> BASIC-Statement FOR ($81) <===
07723  5df9
07724  5df9  a9 80      for     lda #%10000000
07725  5dfb  85 12               sta sublfg        INT Var. sperren
07726  5dfd
07727  5dfd  20 c6 53            jsr let           BASIC-Statement LET ($88)
07728  5e00
07729  5e00  ===> Test FOR mit gleicher Var. auf BASIC Stapel <===
07730  5e00
07731  5e00  a9 81               lda #_fort        FOR Token
07732  5e02  20 aa 4f            jsr search        Token (ac) in BASIC-Stapel suchen
07733  5e05  f0 08               beq for010        auf Stapel gefunden
07734  5e07
07735  5e07  ===> FOR Parameter auf BASIC Stapel legen <===
07736  5e07
07737  5e07  a9 12               lda #_lnfor       Stapelbytes FOR
07738  5e09  20 fe 4f            jsr getstk        prüfen auf Platz im Stapel
07739  5e0c
07740  5e0c  20 47 50            jsr movtos        Kopie Stapelzeiger nach Suchzeiger
07741  5e0f
07742  5e0f
07743  5e0f  ===> FOR Parameter auf BASIC Stapel <===
07744  5e0f
07745  5e0f  20 50 50    for010  jsr movfnd        Kopie Suchzeiger nach Stapelzeiger
07746  5e12  20 a2 52            jsr datan         txtptr hinter FOR Statement
07747  5e15  98                  tya               Zeiger auf Trennzeichen
07748  5e16
07749  5e16  ===> txtptr Offset 17/16 <===
07750  5e16
07751  5e16  a0 11               ldy #_lnfor-1     Zeiger in BASIC Stapel
07752  5e18  18                  clc
07753  5e19  65 3d               adc txtptr        ADDRL chrget-Zeiger
07754  5e1b  91 7d               sta (tos),y       ADDRL txtptr
07755  5e1d
07756  5e1d  a5 3e               lda txtptr+1      ADDRH chrget-Zeiger
07757  5e1f  69 00               adc #0
07758  5e21  88                  dey
07759  5e22  91 7d               sta (tos),y       ADDRH txtptr
07760  5e24
07761  5e24  ===> Zeilennummer Offset 14/15 <===
07762  5e24
07763  5e24  a5 3c               lda curlin+1      High-Byte akt. Zeilennummer
07764  5e26  88                  dey
07765  5e27  91 7d               sta (tos),y       auf BASIC Stapel
07766  5e29
07767  5e29  a5 3b               lda curlin        Low-Byte akt. Zeilennummer
07768  5e2b  88                  dey
07769  5e2c  91 7d               sta (tos),y       auf BASIC Stapel
07770  5e2e
07771  5e2e  a9 a4               lda #_tot         TO Token
07772  5e30  20 5e 79            jsr synchr        Test auf Zeichen in ac, sonst Fehler
07773  5e33
07774  5e33  20 da 77            jsr chknum        Test num. VAR, sonst type mism.
07775  5e36  20 d7 77            jsr frmnum        folg. num. Variable nach FAC
07776  5e3907777  5e39
```

```
07777  5e39  ===> TO Wert in Variablenformat wandeln <===
07778  5e39
07779  5e39  a5 68            lda facsgn       FAC: SG
07780  5e3b  09 7f            ora #%01111111
07781  5e3d  25 64            and facho        FAC: M4
07782  5e3f  85 64            sta facho        FAC: M4
07783  5e41
07784  5e41  ===> TO Wert Offset 9-13 <===
07785  5e41
07786  5e41  a2 04            ldx #4
07787  5e43  a0 0d            ldy #13
07788  5e45  b5 63      for020 lda facexp,x     FAC
07789  5e47  91 7d            sta (tos),y      auf BASIC Stapel
07790  5e49
07791  5e49  ca               dex              Zähler für FAC Bytes
07792  5e4a  88               dey              Zeiger in BASIC Stapel
07793  5e4b
07794  5e4b  ************************  BUG  ********************************
07795  5e4b  *  xr und yr müßten in umgekehrter Reihenfolge dekrementiert *
07796  5e4b  *  werden.                                                    *
07797  5e4b  ***************************************************************
07798  5e4b
07799  5e4b  10 f8            bpl for020
07800  5e4d
07801  5e4d  ===> Ersatzwert für STEP (=1) nach FAC <===
07802  5e4d
07803  5e4d  a9 9c            lda #<fone       GK-Konstante 1.0
07804  5e4f  a0 89            ldy #>fone       GK-Konstante 1.0
07805  5e51  20 d4 8b         jsr movfm        RAM Var. ac/yr nach FAC
07806  5e54
07807  5e54  20 86 03         jsr chrgot       letztes Zeichen erneut nach ac
07808  5e57  c9 a9            cmp #_stept      STEP Token
07809  5e59  d0 06            bne for030
07810  5e5b
07811  5e5b  ===> STEP Wert nach FAC <===
07812  5e5b
07813  5e5b  20 80 03         jsr chrget       nächstes Zeichen nach ac
07814  5e5e  20 d7 77         jsr frmnum       folg. num. Variable nach FAC
07815  5e61  20 57 8c   for030 jsr sign         Flag für FAC NULL, PLUS, MINUS nach
                                               ac
07816  5e64
07817  5e64  48               pha              ,Flag Byte
07818  5e65  20 47 8c         jsr round        FAC runden
07819  5e68  68               pla              Flag Byte
07820  5e69
07821  5e69  ===> Flag Byte und STEP Wert auf BASIC Stapel, Offset 3-8 <===
07822  5e69
07823  5e69  a0 08            ldy #8
07824  5e6b  a2 05            ldx #5
07825  5e6d  91 7d      for040 sta (tos),y      ADDRL BASIC-Stapel Zeiger
07826  5e6f  b5 62            lda expsgn,x     FAC
07827  5e71  88               dey
07828  5e72  ca               dex
07829  5e73  10 f8            bpl for040
07830  5e75
07831  5e75
```

```
07831  5e75  ===> Zeiger auf FOR Variable auf BASIC Stapel, Offset 1-2 <===
07832  5e75
07833  5e75  a5 4c           lda lstpnt+1
07834  5e77  91 7d           sta (tos),y      ADDRL BASIC-Stapel Zeiger
07835  5e79
07836  5e79  a5 4b           lda lstpnt
07837  5e7b  88              dey
07838  5e7c  91 7d           sta (tos),y      ADDRL BASIC-Stapel Zeiger
07839  5e7e
07840  5e7e  ===> FOR Token auf BASIC Stapel <===
07841  5e7e
07842  5e7e  a9 81           lda #_fort       FOR Token
07843  5e80  88              dey
07844  5e81  91 7d           sta (tos),y      ADDRL BASIC-Stapel Zeiger
07845  5e83
07846  5e83  60              rts
07847  5e84
07848  5e84
07849  5e84  ===> Behandlung »?syntax error« <===
07850  5e84
07851  5e84  4c 6c 79   nrange jmp snerr      Behandlung »?syntax error«
07852  5e87
07853  5e87
07854  5e87  ===> BASIC-Statement DELETE ($f7) <===
07855  5e87
07856  5e87  20 f0 84   delete jsr errind     Test Direktmodus, sonst Fehler
07857  5e8a
07858  5e8a  20 86 03         jsr chrgot      letztes Zeichen erneut nach ac
07859  5e8d  f0 f5            beq nrange      DELETE ohne Bereich, »?syntax error«
07860  5e8f
07861  5e8f  20 fb 5e         jsr range       Zeilenbereich einlesen
07862  5e92
07863  5e92  a5 61            lda lowtr       ADDRL von Zeile
07864  5e94  a6 62            ldx lowtr+1     ADDRH von Zeile
07865  5e96
07866  5e96  85 26            sta index2      ADDRL von Zeile
07867  5e98  86 27            stx index2+1    ADDRH von Zeile
07868  5e9a
07869  5e9a  20 64 50         jsr fndlin      Adr. bis Zeile nach lowtr
07870  5e9d  90 15            bcc del300      DELETE bis PGM Ende
07871  5e9f
07872  5e9f  a0 01            ldy #1          Zeiger auf ADDRH Link
07873  5ea1  20 ec 42         jsr indlo0      LDA (lowtr),y aus PGM-RAM
07874  5ea4  88              dey
07875  5ea5  aa              tax             ADDRH Link
07876  5ea6  d0 05           bne noteos      Löschzeiger hinter bis Zeile
07877  5ea8
07878  5ea8  20 ec 42         jsr indlo0      ADDRL Link
07879  5eab  f0 07            beq del300      DELETE bis PGM Ende
07880  5ead
07881  5ead
07882  5ead  ===> Löschzeiger hinter bis Zeile <===
07883  5ead
07884  5ead  20 ec 42   noteos jsr indlo0     ADDRL
07885  5eb0  85 61            sta lowtr       ADDRL bis
07886  5eb2  86 62            stx lowtr+1     ADDRH bis
07887  5eb4
07888  5eb4
```

```
07889  5eb4
07889  5eb4  ===> Test von Zeile kleiner bis Zeile <===
07890  5eb4
07891  5eb4  a5 26      del300 lda index2       ADDRL von Zeile
07892  5eb6  38               sec
07893  5eb7  e5 61             sbc lowtr        ADDRL bis Zeile
07894  5eb9  aa               tax              Differenz ADDRL
07895  5eba
07896  5eba  a5 27             lda index2+1     ADDRH von Zeile
07897  5ebc  e5 62             sbc lowtr+1      ADDRH bis Zeile
07898  5ebe  a8               tay              Differenz ADDRH
07899  5ebf
07900  5ebf  b0 24             bcs notdel       nicht kleiner, nicht DELETE
07901  5ec1
07902  5ec1  ===> txttop nachführen <===
07903  5ec1
07904  5ec1  8a               txa              Differenz ADDRL
07905  5ec2  18               clc
07906  5ec3  6d 10 12          adc txttop       ADDRL Obergrenze akt. PGM
07907  5ec6  8d 10 12          sta txttop       ADDRL Obergrenze akt. PGM
07908  5ec9
07909  5ec9  98               tya              Differenz ADDRH
07910  5eca  6d 11 12          adc txttop+1     ADDRH Obergrenze akt. PGM
07911  5ecd  8d 11 12          sta txttop+1     ADDRH Obergrenze akt. PGM
07912  5ed0
07913  5ed0  ===> DELETE Verschiebe Schleife <===
07914  5ed0
07915  5ed0  a0 00             ldy #0
07916  5ed2  20 ec 42   del500 jsr indlo0
07917  5ed5  91 26             sta (index2),y
07918  5ed7  c8               iny              Seitenzeiger
07919  5ed8  d0 f8             bne del500
07920  5eda
07921  5eda  e6 62             inc lowtr+1      ADDRH Lesezeiger
07922  5edc  e6 27             inc index2+1     ADDRH Schreibzeiger
07923  5ede
07924  5ede  ===> Test Ende Verschiebe Schleife <===
07925  5ede
07926  5ede  ad 11 12          lda txttop+1     ADDRH Obergrenze akt. PGM
07927  5ee1  c5 27             cmp index2+1
07928  5ee3  b0 ed             bcs del500
07929  5ee5
07930  5ee5
07931  5ee5  ===> Abschluß DELETE, Zeiger nachführen <===
07932  5ee5
07933  5ee5  20 4f 4f   notdel jsr lnkprg       alle Linkadressen neu berechnen
07934  5ee8
07935  5ee8  a5 24             lda index1
07936  5eea  a6 25             ldx index1+1
07937  5eec
07938  5eec  18               clc
07939  5eed  69 02             adc #2
07940  5eef  8d 10 12          sta txttop       ADDRL Obergrenze akt. PGM
07941  5ef2  90 01             bcc not500
07942  5ef4  e8               inx
07943  5ef5  8e 11 12   not500 stx txttop+1     ADDRH Obergrenze akt. PGM
07944  5ef8
07945  5ef8  4c 37 4d          jmp ready        Ausgabe »ready.« und ready-Status
```

```
07946  5efb
07947  5efb
07948  5efb  ===> Zeilenbereich einlesen <===
07949  5efb
07950  5efb  f0 12      range  beq rng100       Trennzeichen: 0-65535
07951  5efd  90 10             bcc rng100       dez.Ziffer
07952  5eff
07953  5eff  c9 ab             cmp #_minut      MINUS Token
07954  5f01  d0 0c             bne rng100
07955  5f03
07956  5f03  ===> »?syntax error«, wenn Trennzeichen hinter MINUS <===
07957  5f03
07958  5f03  a0 01             ldy #1           Zeiger hinter MINUS
07959  5f05  20 c9 03          jsr indtxt       LDA (txtptr),y aus PGM-RAM
07960  5f08  f0 27             beq rngerr       Zeilenende
07961  5f0a
07962  5f0a  c9 3a             cmp #':'
07963  5f0c  f0 23             beq rngerr       Trennzeichen
07964  5f0e  38                sec
07965  5f0f
07966  5f0f  20 a0 50   rng100 jsr linget       von Zeile nach linnum
07967  5f12  20 64 50          jsr fndlin       Adr. von Zeile nach lowtr/+1
07968  5f15
07969  5f15  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
07970  5f18  f0 0c             beq rng200       Zeilenende
07971  5f1a
07972  5f1a  c9 ab             cmp #_minut      MINUS Token
07973  5f1c  d0 13             bne rngerr       Behandlung »?syntax errror«
07974  5f1e
07975  5f1e  ===> bis Zeile einlesen <===
07976  5f1e
07977  5f1e  20 80 03          jsr chrget       nächstes Zeichen nach ac
07978  5f21  20 a0 50          jsr linget       Zeilennr. nach linnum
07979  5f24  d0 0b             bne rngerr       nicht Zeilenende
07980  5f26
07981  5f26  a5 0a      rng200 lda endchr       Abbruchzeichen
07982  5f28  d0 06             bne rngrts
07983  5f2a
07984  5f2a  ===> bis Zeile mit 65535 vorbesetzen <===
07985  5f2a
07986  5f2a  a9 ff             lda #>65535
07987  5f2c  85 16             sta linnum
07988  5f2e  85 17             sta linnum+1
07989  5f30
07990  5f30  60         rngrts rts
07991  5f31
07992  5f31
07993  5f31  ===> Behandlung »?syntax error« <===
07994  5f31
07995  5f31  4c 6c 79   rngerr jmp snerr        Behandlung »?syntax errror«
07996  5f34
07997  5f34
07998  5f34  ===> BASIC-Statement PUDEF ($dd) <===
07999  5f34
08000  5f34  20 7b 87   puctrl jsr sav13        Ausdruck in Arbeitsstring, frestr
08001  5f37  a8                tay              Stringlänge
08002  5f38  88                dey              Zeiger in String
08003  5f39
```

```
08004  5f39  c0 04            cpy #4
08005  5f3b  90 03            bcc pulop       PRINT USING Definitionen speichern
08006  5f3d
08007  5f3d  ===> Behandlung »?illegal quantity error« <===
08008  5f3d
08009  5f3d  4c 28 7d         jmp fcerr       Behandlung »?illegal quantity error«
08010  5f40
08011  5f40
08012  5f40  ===> PRINT USING Definitionen speichern <===
08013  5f40
08014  5f40  20 b7 03   pulop  jsr ind1_1      LDA (index1),y aus VAR-RAM
08015  5f43
08016  5f43  8d 03 ff         sta lcrc        ROM / PGM-RAM einschalten
08017  5f46
08018  5f46  99 04 12         sta pufill,y    PRINT USING: Füllzeichen
08019  5f49  88               dey
08020  5f4a  10 f4            bpl pulop       PRINT USING Definitionen speichern
08021  5f4c  60               rts
08022  5f4d
08023  5f4d
08024  5f4d  ===> BASIC-Statement TRAP ($d7) <===
08025  5f4d
08026  5f4d  20 d9 84   trap   jsr errdir      Test Direktmodus, Behandl. »?illegal
                                               direct error«
08027  5f50
08028  5f50  20 86 03         jsr chrgot      letztes Zeichen erneut nach ac
08029  5f53  f0 07            beq trap1       TRAP-Funktion ausschalten
08030  5f55
08031  5f55  ===> Zeilennummer für TRAP einlesen <===
08032  5f55
08033  5f55  20 12 88         jsr getwrd      16 Bit INT nach linnum
08034  5f58  8c 0b 12         sty trapno      low Zeilennr. TRAP Routine
08035  5f5b  2c               .byte _skip2
08036  5f5c
08037  5f5c
08038  5f5c  ===> TRAP-Funktion ausschalten <===
08039  5f5c
08040  5f5c  a9 ff      trap1 lda #$ff         Flag für TRAP OFF
08041  5f5e  8d 0c 12         sta trapno+1    high Zeilennr. TRAP Routine
08042  5f61  60               rts
08043  5f62
08044  5f62
08045  5f62  ===> BASIC-Statement RESUME ($d6) <===
08046  5f62
08047  5f62  20 d9 84   resume jsr errdir      Test Direktmodus, Behandl. »?illegal
                                               direct error«
08048  5f65
08049  5f65  ===> Test auf Fehlerzeile <===
08050  5f65
08051  5f65  ae 0a 12         ldx errlin+1    High-Byte Zeilennummer für RESUME
08052  5f68  e8               inx
08053  5f69  f0 70            beq rescnt      Behandlung »can't resume error«
08054  5f6b
08055  5f6b  20 86 03         jsr chrgot      letztes Zeichen erneut nach ac
08056  5f6e  f0 47            beq resswp      errlin nach curlin, errtxt nach
                                               txtptr
08057  5f70  90 3a            bcc resnum      RESUME mit Zeilennummer
08058  5f72
```

```
08059  5f72  c9 82           cmp #_nextt      NEXT Token
08060  5f74  d0 62           bne ressnr       Behandlung »?syntax error«
08061  5f76
08062  5f76
08062  5f76  ===> RESUME NEXT <===
08063  5f76
08064  5f76  20 b7 5f        jsr resswp       errlin nach curlin, errtxt nach
                                              txtptr
08065  5f79
08066  5f79  a0 00           ldy #0           Zeiger hinter Fehlerstelle
08067  5f7b  20 c9 03        jsr indtxt       LDA (txtptr),y aus PGM-RAM
08068  5f7e  d0 26           bne resum2       nicht Zeilenende
08069  5f80
08070  5f80  ===> Zeilenende <===
08071  5f80
08072  5f80  c8              iny              Zeiger ADDRL Link
08073  5f81  20 c9 03        jsr indtxt       LDA (txtptr),y aus PGM-RAM
08074  5f84  d0 09           bne resum1       curlin/txtptr nächste Zeile
08075  5f86
08076  5f86  c8              iny              Zeiger ADDRH Link
08077  5f87  20 c9 03        jsr indtxt       LDA (txtptr),y aus PGM-RAM
08078  5f8a  d0 03           bne resum1       curlin/txtptr nächste Zeile
08079  5f8c
08080  5f8c  ===> PGM Ende <===
08081  5f8c
08082  5f8c  4c 37 4d        jmp ready        Ausgabe »ready.« und ready-Status
08083  5f8f
08084  5f8f
08085  5f8f  ===> curlin/txtptr nächste Zeile <===
08086  5f8f
08087  5f8f  a0 03     resum1 ldy #3           Zeiger Low-Byte Zeilennr.
08088  5f91  20 c9 03        jsr indtxt       LDA (txtptr),y aus PGM-RAM
08089  5f94  85 3b           sta curlin       Low-Byte akt. Zeilennummer
08090  5f96
08091  5f96  c8              iny              Zeiger High-Byte Zeilennr.
08092  5f97  20 c9 03        jsr indtxt       LDA (txtptr),y aus PGM-RAM
08093  5f9a  85 3c           sta curlin+1     High-Byte akt. Zeilennummer
08094  5f9c
08095  5f9c  98              tya              Zeiger High-Byte Zeilennr.
08096  5f9d  18              clc
08097  5f9e  65 3d           adc txtptr       ADDRL chrget-Zeiger
08098  5fa0  85 3d           sta txtptr       ADDRL chrget-Zeiger
08099  5fa2  90 02           bcc resum2       txtptr auf nächstes Trennzeichen
08100  5fa4  e6 3e           inc txtptr+1     ADDRH chrget-Zeiger
08101  5fa6
08102  5fa6
08103  5fa6  ===> txtptr auf nächstes Trennzeichen <===
08104  5fa6
08105  5fa6  20 80 03   resum2 jsr chrget      txtptr auf nächstes Trennzeichen
08106  5fa9  4c 8f 52        jmp data         BASIC-Statement DATA ($83)
08107  5fac
08108  5fac
08109  5fac  ===> RESUME mit Zeilennummer <===
08110  5fac
08111  5fac  20 12 88   resnum jsr getwrd     16 Bit INT nach linnum
08112  5faf  85 17           sta linnum+1     ****** ? BUG ? ******
08113  5fb1
08114  5fb1  20 c6 5f        jsr resend       Flags setzen RESUME Ende
```

```
08115  5fb4  4c fb 59          jmp lukall       Zeilennr. < akt. suchen
08116  5fb7
08117  5fb7
08118  5fb7
08118  5fb7  ===> errlin nach curlin, errtxt nach txtptr <===
08119  5fb7
08120  5fb7  a2 01      resswp ldx #1
08121  5fb9  bd 09 12   rswplp lda errlin,x     Low-Byte Zeilennummer für RESUME
08122  5fbc  95 3b             sta curlin,x     Low-Byte akt. Zeilennummer
08123  5fbe
08124  5fbe  bd 0e 12          lda errtxt,x     ADDRL txtptr bei Fehler
08125  5fc1  95 3d             sta txtptr,x     ADDRL chrget-Zeiger
08126  5fc3
08127  5fc3  ca                dex
08128  5fc4  10 f3             bpl rswplp
08129  5fc6
08130  5fc6
08131  5fc6  ===> Flags setzen RESUME Ende <===
08132  5fc6
08133  5fc6  a2 ff      resend ldx #$ff
08134  5fc8  8e 08 12          stx errnum       Nummer Fehlertext
08135  5fcb
08136  5fcb  8e 09 12          stx errlin       Low-Byte Zeilennummer für RESUME
08137  5fce  8e 0a 12          stx errlin+1     High-Byte Zeilennummer für RESUME
08138  5fd1
08139  5fd1  ae 0d 12          ldx tmptrp       high Zeilennr. TRAP Routine bei TRAP
08140  5fd4  8e 0c 12          stx trapno+1     high Zeilennr. TRAP Routine
08141  5fd7
08142  5fd7  60                rts
08143  5fd8
08144  5fd8
08145  5fd8  ===> Behandlung »?syntax error« <===
08146  5fd8
08147  5fd8  4c 6c 79   ressnr jmp snerr        Behandlung »?syntax error«
08148  5fdb
08149  5fdb
08150  5fdb  ===> Behandlung »can't resume error« <===
08151  5fdb
08152  5fdb  a2 1f      rescnt ldx #_ercr
08153  5fdd  4c 3c 4d          jmp error        ind. Behandlung Fehlermeldung
08154  5fe0
08155  5fe0
08156  5fe0  ===> BASIC-Statement DO ($eb) <===
08157  5fe0
08158  5fe0  a0 01      do     ldy #1
08159  5fe2
08160  5fe2
08161  5fe2  ===> chrget-Zeiger, Zeilennr. retten <===
08162  5fe2
08163  5fe2  b9 3d 00   dolop1 lda txtptr,y     ADDRL chrget-Zeiger
08164  5fe5  99 14 12          sta tmptxt,y     Kopie ADDRL chrget-Zeiger
08165  5fe8
08166  5fe8  b9 3b 00          lda curlin,y     Low-Byte akt. Zeilennummer
08167  5feb  99 16 12          sta tmplin,y     Kopie Low-Byte Zeilennummer
08168  5fee
08169  5fee  88                dey
08170  5fef  10 f1             bpl dolop1       chrget-Zeiger, Zeilennr. retten
08171  5ff1
```

```
08172  5ff1  20 86 03          jsr chrgot      letztes Zeichen erneut nach ac
08173  5ff4  f0 1c             beq doyes       Zeilenende
08174  5ff6
08175  5ff6  c9 fc             cmp #_untlt     UNTIL-Token
08176  5ff8  f0 11             beq do10        DO UNTIL
08177  5ffa
08178  5ffa  c9 fd             cmp #_whilt     WHILE-Token
08179  5ffc  d0 43             bne xtsner      Behandlung »?syntax error«
08180  5ffe
08181  5ffe  ===> DO WHILE <===
08182  5ffe
08183  5ffe  20 db 60          jsr frmjmp      chrget, folg. Ausdr. auswerten
08184  6001
08185  6001  a5 63             lda facexp      FAC: EX
08186  6003  d0 0d             bne doyes       <>0, Ausdruck wahr, weiter
08187  6005
08188  6005
08189  6005  ===> LOOP verlassen <===
08190  6005
08191  6005  20 86 03   don0    jsr chrgot      letztes Zeichen erneut nach ac
08192  6008  4c 47 60          jmp fnd010
08193  600b
08194  600b
08195  600b  ===> DO UNTIL <===
08196  600b
08197  600b  20 db 60   do10    jsr frmjmp      chrget, folg. Ausdr. auswerten
08198  600e
08199  600e  a5 63             lda facexp      FAC: EX
08200  6010  d0 f3             bne don0        <>0, Ausdruck wahr, Ende
08201  6012
08202  6012
08203  6012  ===> DO Parameter auf BASIC-Stapel <===
08204  6012
08205  6012  a9 05      doyes  lda #_lngos     Stapelbytes GOSUB
08206  6014  20 fe 4f          jsr getstk      Platz in BASIC-Stapel
08207  6017
08208  6017  8d 03 ff          sta lcrc        ROM / PGM-RAM einschalten
08209  601a
08210  601a  ===> DO Token, chrget-Zgr., Zeilennr. auf BASIC-Stapel <===
08211  601a
08212  601a  a0 04             ldy #_lngos-1   Zähler für 5 Zeichen
08213  601c  ad 15 12          lda tmptxt+1    Kopie ADDRH chrget-Zeiger
08214  601f  91 7d             sta (tos),y     auf BASIC-Stapel
08215  6021
08216  6021  88                dey
08217  6022  ad 14 12          lda tmptxt      Kopie ADDRL chrget-Zeiger
08218  6025  91 7d             sta (tos),y     auf BASIC-Stapel
08219  6027
08220  6027  88                dey
08221  6028  ad 17 12          lda tmplin+1    Kopie High-Byte Zeilennummer
08222  602b  91 7d             sta (tos),y     auf BASIC-Stapel
08223  602d
08224  602d  88                dey
08225  602e  ad 16 12          lda tmplin      Kopie Low-Byte Zeilennummer
08226  6031  91 7d             sta (tos),y     auf BASIC-Stapel
08227  6033
08228  6033  88                dey
08229  6034  a9 eb             lda #_dot       DO Token
```

```
08230  6036  91 7d            sta (tos),y      auf BASIC-Stapel
08231  6038
08232  6038  60               rts
08233  6039
08234  6039
08235  6039
08235  6039  ===> BASIC-Statement EXIT ($ed) <===
08236  6039
08237  6039  20 9b 60   exit   jsr popdgo      DO Token vom BASIC-Stapel
08238  603c
08239  603c  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
08240  603f  f0 06             beq fnd010
08241  6041
08242  6041  ===> Behandlung »?syntax error« <===
08243  6041
08244  6041  4c 6c 79   xtsner jmp snerr       Behandlung »?syntax error«
08245  6044
08246  6044
08247  6044  ===> chrget-Zgr. hinter LOOP setzen <===
08248  6044
08249  6044  20 80 03   fndend jsr chrget      nächstes Zeichen nach ac
08250  6047  f0 17      fnd010 beq fnd50       Behandlung Trennzeichen
08251  6049
08252  6049  c9 ec             cmp #_loopt      LOOP Token
08253  604b  f0 3a             beq fnd60        LOOP-Zeile überlesen
08254  604d
08255  604d  c9 22             cmp #'"'
08256  604f  f0 0a             beq fnd40        Text in "" überlesen, Test Trennz.
08257  6051
08258  6051  c9 eb             cmp #_dot        DO Token
08259  6053  d0 ef             bne fndend       chrget-Zgr. hinter LOOP setzen
08260  6055
08261  6055  ===> LOOP innerhalb LOOP überlesen <===
08262  6055
08263  6055  20 44 60          jsr fndend       chrget-Zgr. hinter LOOP setzen
08264  6058  4c 05 60          jmp don0         LOOP verlassen
08265  605b
08266  605b
08267  605b  ===> Text in "" überlesen, Test Trennz. <===
08268  605b
08269  605b  20 7c 53   fnd40  jsr unquot       Text in "" überlesen
08270  605e  d0 e4             bne fndend       chrget-Zgr. hinter LOOP setzen
08271  6060
08272  6060
08273  6060  ===> Behandlung Trennzeichen <===
08274  6060
08275  6060  c9 3a      fnd50  cmp #':'         Trennzeichen für BASIC-Befehle
08276  6062  f0 e0             beq fndend       noch nicht Zeilenende
08277  6064
08278  6064  24 7f             bit runmod       Test Bit7, im Direktmodus
08279  6066  10 42             bpl fnderr       Behandlung »?loop not found error«
08280  6068
08281  6068  a0 02             ldy #2           Zeiger auf ADDRH Link
08282  606a  20 c9 03          jsr indtxt       ADDRH Link (0=PGM-Ende)
08283  606d  f0 3b             beq fnderr       Behandlung »?loop not found error«
08284  606f
```

```
08285  606f  ===> nächste Zeilennummer einlesen <===
08286  606f
08287  606f  c8              iny
08288  6070  20 c9 03        jsr indtxt      LDA (txtptr),y aus PGM-RAM
08289  6073  85 3b           sta curlin      Low-Byte akt. Zeilennummer
08290  6075
08291  6075  c8              iny
08292  6076  20 c9 03        jsr indtxt      LDA (txtptr),y aus PGM-RAM
08293  6079  85 3c           sta curlin+1    High-Byte akt. Zeilennummer
08294  607b
08295  607b  ===> chrget-Zeiger nachführen <===
08296  607b
08297  607b  98              tya
08298  607c  18              clc
08299  607d  65 3d           adc txtptr      ADDRL chrget-Zeiger
08300  607f  85 3d           sta txtptr      ADDRL chrget-Zeiger
08301  6081
08302  6081  90 c1           bcc fndend      weiter suchen
08303  6083  e6 3e           inc txtptr+1    ADDRH chrget-Zeiger
08304  6085  d0 bd           bne fndend      immer
08305  6087
08306  6087
08307  6087  4c 8f 52   fnd60  jmp data       LOOP-Zeile überlesen
08308  608a
08309  608a
08310  608a  ===> BASIC-Statement LOOP (ec) <===
08311  608a
08312  608a  f0 35      loop   beq popngo     Trennzeichen gefunden
08313  608c
08314  608c  c9 fd           cmp #_whilt     WHILE Token
08315  608e  f0 2c           beq loop10      LOOP WHILE
08316  6090
08317  6090  c9 fc           cmp #_untlt     UNTIL Token
08318  6092  d0 ad           bne xtsner      Behandlung »?syntax error«
08319  6094
08320  6094  ===> LOOP UNTIL <===
08321  6094
08322  6094  20 db 60        jsr frmjmp      chrget, folg. Ausdr. auswerten
08323  6097
08324  6097  a5 63           lda facexp      FAC: EX
08325  6099  f0 26           beq popngo      =0, Ausdruck nicht wahr
08326  609b
08327  609b
08328  609b  ===> DO Token vom BASIC-Stapel <===
08329  609b
08330  609b  a9 eb    popdgo lda #_dot       DO Token
08331  609d  20 aa 4f        jsr search      im BASIC-Stapel suchen
08332  60a0  d0 15           bne poperr      Behandlung »?loop without do error«
08333  60a2
08334  60a2  20 50 50        jsr movfnd      Kopie Suchzeiger nach Stapelzeiger
08335  60a5
08336  60a5  a0 05           ldy #_lngos     Stapelbytes GOSUB
08337  60a7  4c 59 50        jmp rlsstk      BASIC-Stapel Zeiger um yr erhöhen
08338  60aa
08339  60aa
08340  60aa  ===> Behandlung »?loop not found error« <===
08341  60aa
08342  60aa  ad 16 12   fnderr lda tmplin     Kopie Low-Byte Zeilennummer
```

```
08343  60ad  ae 17 12          ldx tmplin+1     Kopie High-Byte Zeilennummer
08344  60b0
08345  60b0  85 3b             sta curlin       Low-Byte akt. Zeilennummer
08346  60b2  86 3c             stx curlin+1     High-Byte akt. Zeilennummer
08347  60b4
08348  60b4  a2 20             ldx #_erlnf
08349  60b6  2c                .byte _skip2
08350  60b7
08351  60b7
08352  60b7
08352  60b7  ===> Behandlung »?loop without do error« <===
08353  60b7
08354  60b7  a2 21       poperr ldx #_erlwd
08355  60b9  4c 3c 4d           jmp error        ind. Behandlung Fehlermeldung
08356  60bc
08357  60bc
08358  60bc  ===> LOOP WHILE <===
08359  60bc
08360  60bc  20 db 60    loop10 jsr frmjmp       chrget, folg. Ausdr. auswerten
08361  60bf  f0 da              beq popdgo       Ausdruck falsch, Ende Schleife
08362  60c1
08363  60c1
08364  60c1  ===> Start DO Schleife <===
08365  60c1
08366  60c1  20 9b 60    popngo jsr popdgo       DO Token vom BASIC-Stapel
08367  60c4
08368  60c4  ===> chrget-Zeiger, Zeilennr. vom Stapel <===
08369  60c4
08370  60c4  88                 dey
08371  60c5  b1 3f              lda (fndpnt),y   ADDRH chrget-Zeiger auf DO
08372  60c7  85 3e              sta txtptr+1     ADDRH chrget-Zeiger
08373  60c9
08374  60c9  88                 dey
08375  60ca  b1 3f              lda (fndpnt),y   ADDRL chrget-Zeiger auf DO
08376  60cc  85 3d              sta txtptr       ADDRL chrget-Zeiger
08377  60ce
08378  60ce  88                 dey
08379  60cf  b1 3f              lda (fndpnt),y   High-Byte Zeilennummer
08380  60d1  20 3b a8           jsr retpat       vom Stapel, (Direkt-) Modus
                                                 eintragen
08381  60d4
08382  60d4  b1 3f              lda (fndpnt),y   Low-Byte Zeilennummer
08383  60d6  85 3b              sta curlin       Low-Byte akt. Zeilennummer
08384  60d8
08385  60d8  4c e0 5f           jmp do           BASIC-Statement DO ($eb)
08386  60db
08387  60db
08388  60db  ===> chrget, folg. Ausdr. auswerten <===
08389  60db
08390  60db  20 80 03    frmjmp jsr chrget       nächstes Zeichen nach ac
08391  60de  4c ef 77           jmp frmevl       folgenden Ausdruck auswerten
08392  60e1
08393  60e1
08394  60e1  ===> Fkt.Tasten Text definieren <===
08395  60e1
08396  60e1  20 f4 87    chgkey jsr getbyt       Nummer Fkt.-Taste
08397  60e4  ca                 dex
08398  60e5  e0 08              cpx #8
```

```
08399  60e7  90 03            bcc chgky1
08400  60e9
08401  60e9  ===> Behandlung »?illegal quantity error« <===
08402  60e9
08403  60e9  4c 28 7d         jmp fcerr        Behandlung »?illegal quantity error«
08404  60ec
08405  60ec
08406  60ec  86 77      chgky1 stx mltus1       Nummer Fkt.Taste -1
08407  60ee
08408  60ee  20 5c 79         jsr chkcom       Test auf folg. »Komma« sonst Fehler
08409  60f1  20 7b 87         jsr sav13        Ausdruck in Arbeitsstring, frestr
08410  60f4
08411  60f4  a8               tay              Länge Fkt.Tasten-Text
08412  60f5
08413  60f5  a9 01            lda #_varbk      VAR RAM
08414  60f7  85 26            sta index1+2     Bank für keyset (OPSYS)
08415  60f9
08416  60f9  a9 24            lda #<index1     LDAVEC für keyset (OPSYS)
08417  60fb  a6 77            ldx mltus1       Nummer Fkt.Taste -1
08418  60fd  e8               inx              Nummer Fkt.Taste für keyset (OPSYS)
08419  60fe
08420  60fe  20 45 a8         jsr mapio        I/O-Chips einblenden
08421  6101  20 65 ff         jsr kpfkey       Funktionstasten programmieren
                                                (Sprungv.)
08422  6104  b0 01            bcs omerr3       Länge Texte aller Fkt.Tasten > 255
08423  6106
08424  6106  60               rts
08425  6107
08426  6107
08427  6107  ===> Behandlung »?out of memory error« <===
08428  6107
08429  6107  4c 3a 4d   omerr3 jmp omerr       Behandlung »?out of memory error«
08430  610a
08431  610a
08432  610a  ===> BASIC-Statement KEY ($f9) <===
08433  610a
08434  610a  f0 15       key    beq lstky0       Trennz., LIST Fkt.Tasten
08435  610c
08436  610c  ===> Test auf KEY ON / OFF (nicht implem.) <===
08437  610c
08438  610c  c9 91            cmp #_ont        ON Token
08439  610e  f0 0e            beq badco1       Behandlung »?unimplemented command
                                                error«
08440  6110
08441  6110  c9 fe            cmp #_ecmdt      ESC COMMAND Token
08442  6112  d0 cd            bne chgkey       Fkt.Tasten Text definieren
08443  6114
08444  6114  20 80 03         jsr chrget       nächstes Zeichen nach ac
08445  6117  c9 24            cmp #_offt       OFF Token
08446  6119  f0 03            beq badco1       Behandlung »?unimplemented command
                                                error«
08447  611b
08448  611b  ===> Behandlung »?syntax error« <===
08449  611b
08450  611b  4c 6c 79         jmp snerr        Behandlung »?syntax error«
08451  611e
08452  611e
```

```
08453  611e  ===> Behandlung »?unimplemented command error« <===
08454  611e
08455  611e  4c 46 48   badco1 jmp badcom       Behandlung »?unimplemented command
                                                error«
08456  6121
08457  6121
08458  6121  ===> alle Keys anzeigen <===
08459  6121
08460  6121  a2 00      lstky0 ldx #0           Zähler Fkt.Tasten (0-7)
08461  6123  a0 00             ldy #0           Zeiger in Texte Fkt.Tasten
08462  6125
08463  6125  e8         lstky1 inx
08464  6126  bd ff 0f          lda pkylen-1,x   Tab. Längen Fkt.Tasten Texte
08465  6129  f0 53             beq lstest       Länge =0, nächste Fkt.Taste
08466  612b
08467  612b  85 78             sta hulp         Zeichenzähler Text Fkt.Taste
08468  612d  86 77             stx mltus1       Nummer Fkt.Taste
08469  612f
08470  612f  ===> Ausgabe »KEY (Nr.), « <===
08471  612f
08472  612f  a2 05             ldx #5           Zähler für 6 Zeichen
08473  6131  bd 2a a8   lstky2 lda preamb,x     Text »key 0,«
08474  6134  ca                dex
08475  6135  d0 02             bne lstky3
08476  6137
08477  6137   xr=0 für »0« aus preamb, log.oder mit Nr. -> ASCII Nr.
08478  6137
08479  6137  05 77             ora mltus1       Nummer Fkt.Taste
08480  6139
08481  6139  20 69 92   lstky3 jsr cbsout       ind. Ausgabe ac auf akt. Kanal
08482  613c
08483  613c  8a                txa              Flags setzen
08484  613d  10 f2             bpl lstky2       Ausgabe nächstes Zeichen
08485  613f
08486  613f
08487  613f  ===> Ausgabe Text akt. Fkt.Taste <===
08488  613f
08489  613f  a2 07             ldx #7           Flag Textanfang
08490  6141
08491  6141
08492  6141  ===> Zeichen aus KEY-Definition holen <===
08493  6141
08494  6141  b9 0a 10   lsloop lda pkydef,y     Zeichen Text Fkt.Taste
08495  6144  c8                iny              Zeiger in Texten
08496  6145
08497  6145  48                pha              Zeichen Text Fkt.Taste
08498  6146  86 79             stx syntmp       Flag
08499  6148
08500  6148  ===> Test auf Steuerz. in Text Fkt.Taste <===
08501  6148
08502  6148  a2 04             ldx #4           5 Steuerzeichen
08503  614a  dd a3 61   lstky5 cmp kychtb,x     Tabelle der KEY-Sonderzeichen
08504  614d  f0 34             beq lstspc       Ausgabe CHR$(
08505  614f
08506  614f  ca                dex
08507  6150  d0 f8             bne lstky5
08508  6152
```

```
08509  6152  ===> LIST Steuerzeichen »+chr$(xx)« <===
08510  6152
08511  6152  a6 79           ldx syntmp       Flag lesen
08512  6154  e0 08           cpx #8
08513  6156  90 07           bcc lstky6       Anfangsspalte, Anführungszeichen
08514  6158  d0 0a           bne lstky7       Ausgabe Text (ASCII)
08515  615a
08516  615a  ===> Ausgabe »+"« nach »+chr$(x)« <===
08517  615a
08518  615a  a9 2b           lda #'+'         Flag =8
08519  615c  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
08520  615f
08521  615f
08522  615f
08522  615f  ===> Ausgabe Anführungszeichen <===
08523  615f
08524  615f  a9 22     lstky6 lda #'"'
08525  6161  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
08526  6164
08527  6164  68         lstky7 pla             Zeichen Text Fkt.Taste
08528  6165  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
08529  6168
08530  6168  a2 09           ldx #9           Flag Ausgabe Text (ASCII)
08531  616a  c6 78     lstnd  dec hulp         Zeichenzähler Text Fkt.Taste
08532  616c  d0 d3           bne lsloop       Zeichen aus KEY-Definition holen
08533  616e
08534  616e  e0 09           cpx #9           Flag Ausgabe Text (ASCII)
08535  6170  90 05           bcc lstky8       Abschluss mit »SHIFT CR«
08536  6172
08537  6172  a9 22           lda #'"'
08538  6174  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
08539  6177
08540  6177
08541  6177  ===> Abschluß mit »SHIFT CR« <===
08542  6177
08543  6177  a9 8d     lstky8 lda #_sftcr     »shift return«
08544  6179  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
08545  617c
08546  617c  a6 77           ldx mltus1       Nummer Fkt.Taste (0-7)
08547  617e  e0 08     lstest cpx #8
08548  6180  d0 a3           bne lstky1
08549  6182
08550  6182  60               rts
08551  6183
08552  6183
08553  6183  ===> Ausgabe CHR$( <===
08554  6183
08555  6183  a6 79     lstspc ldx syntmp       Flag (mit »+« oder ohne)
08556  6185  bd 9a 61  chrlop lda keydat-3,x   Text »+chr$(« für LIST Fkt.Taste
08557  6188  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
08558  618b
08559  618b  ca               dex
08560  618c  e0 03           cpx #3           Zeiger auf »(«
08561  618e  b0 f5           bcs chrlop
08562  6190
08563  6190  68               pla              Steuerzeichen
08564  6191  20 30 a8        jsr prtdec       8-Bit INT nach ASCII auf akt. Kanal
08565  6194
```

```
08566  6194  a9 29          lda #')'
08567  6196  20 69 92       jsr cbsout      ind. Ausgabe ac auf akt. Kanal
08568  6199
08569  6199  a2 08          ldx #8          Flag 8 (nach Ausgabe chr$)
08570  619b  d0 cd          bne lstnd       immer
08571  619d
08572  619d
08573  619d  ===> Text »+chr$(« für LIST Fkt.Taste <===
08574  619d
08575  619d  28 24 52   keydat .byte '($rhc+'
08575  61a0  48 43 2b
08576  61a3
08577  61a3
08578  61a3
08578  61a3  ===> Tabelle der KEY-Sonderzeichen <===
08579  61a3
08580  61a3  22 0d 8d   kychtb .byte '"', _cr, _sftcr, '"', _esc
08580  61a6  22 1b
08581  61a8
08582  61a8                  .end
08583  61a8                  .lib basic.graph_1
08584  61a8  put"@basic.graph_1"
```

```
08586  61a8
08587  61a8
08588  61a8  ===> BASIC-Statement PAINT ($df) <===
08589  61a8
08590  61a8  20 2f 9e   paint  jsr grpcol       Test GRAPHIC, Farbqu. nach colsel
08591  61ab
08592  61ab  a2 04             ldx #4          Zeiger in Koordinatentabelle
08593  61ad  20 52 9e          jsr in2cor      Ersatz-/Start-/Zielkoord. einlesen
08594  61b0  20 f2 9d          jsr dstpos      Zielkoord. werden akt. Koord.
08595  61b3
08596  61b3  20 1c 9e          jsr optzer      opt. 8 Bit INT nach xr, sonst 0
08597  61b6  e0 02             cpx #2          Modus
08598  61b8  90 03             bcc paint2
08599  61ba
08600  61ba  ===> Behandlung »?illegal quantity error« <===
08601  61ba
08602  61ba  4c 28 7d          jmp fcerr       Behandlung »?illegal quantity error«
08603  61bd
08604  61bd
08605  61bd  8a         paint2 txa             Modus
08606  61be  4a                lsr a           Bit 0
08607  61bf  6a                ror a           nach Bit 7
08608  61c0  85 8b             sta stopnb      Flag für paint, Rahmenflag
08609  61c2
08610  61c2   7.......  1 = Modus 1, 0 = Modus 0
08611  61c2
08612  61c2  10 04             bpl pntok1      Punkt in Bit-Map testen
08613  61c4
08614  61c4  a5 83             lda colsel      Farbquelle (0-3)
08615  61c6  f0 07             beq pnterr
08616  61c8
08617  61c8
08618  61c8  ===> Punkt in Bit-Map testen <===
08619  61c8
08620  61c8  20 49 9c   pntok1 jsr readpt      Test auf Punkt xr/yr gesetzt
08621  61cb  b0 02             bcs pnterr      nicht im Graphikbereich
08622  61cd  d0 01             bne pntok2      nicht gesetzt
08623  61cf
08624  61cf  60         pnterr rts             nicht PAINT
08625  61d0
08626  61d0
08627  61d0  ===> PAINT-Stapel einrichten <===
08628  61d0
08629  61d0  20 ea 92   pntok2 jsr garba2      Garbage Collection
08630  61d3  8d 03 ff          sta lcrc        ROM / PGM-RAM einschalten
08631  61d6
08632  61d6  a5 33             lda strend      ADDRL+1 Obergr. ARRAY Tabelle
08633  61d8  85 24             sta index1      ADDRL Zeiger Koordinaten-Stapel
08634  61da
08635  61da  a5 34             lda strend+1    ADDRH+1 Obergr. ARRAY Tabelle
08636  61dc  85 25             sta index1+1    ADDRH Zeiger Koordinaten-Stapel
08637  61de
08638  61de  38                sec
08639  61df  a5 35             lda fretop      ADDRL Zeiger Anfang String-Variable
08640  61e1  e9 03             sbc #3
08641  61e3  85 1b             sta tempst      ADDRL max. Stapel-Bereich
08642  61e5
08643  61e5  a5 36             lda fretop+1    ADDRH Zeiger Anfang String-Variable
```

```
08644  61e7  e9 00           sbc #0
08645  61e9  85 1c           sta tempst+1     ADDRH max. Stapel-Bereich
08646  61eb
08647  61eb
08648  61eb  ===> PAINT in x-Richtung ausführen <===
08649  61eb
08650  61eb  a2 00      ptloop ldx #0
08651  61ed  86 63           stx facexp       Flag für Stapeleintrag li.
08652  61ef  86 64           stx facho        Flag für Stapeleintrag re.
08653  61f1
08654  61f1   7.......  1 = Eintrag, 0 = nicht
08655  61f1
08656  61f1
08657  61f1  ae 33 11  ptdown ldx xypos+2     Absz. low PIXEL CURS.
08658  61f4  d0 03           bne ptdwn2
08659  61f6  ce 34 11        dec xypos+3      Absz. high PIXEL CURS.
08660  61f9  ce 33 11  ptdwn2 dec xypos+2     vermindern
08661  61fc
08662  61fc  20 49 9c        jsr readpt       Test auf Punkt xr/yr gesetzt
08663  61ff  b0 02           bcs ptbotm       nicht im Graphikbereich
08664  6201  d0 ee           bne ptdown       nicht gesetzt
08665  6203
08666  6203  ee 33 11  ptbotm inc xypos+2     Absz. low PIXEL CURS.
08667  6206  d0 03           bne ptfill       PAINT in y-Richtung ausführen
08668  6208  ee 34 11        inc xypos+3      Absz. high PIXEL CURS.
08669  620b
08670  620b
08671  620b  ===> PAINT in y-Richtung ausführen <===
08672  620b
08673  620b  20 19 9c  ptfill jsr plot01      Graphik Punkt setzen
08674  620e
08675  620e  ae 31 11        ldx vwork
08676  6211  d0 03           bne ptfil2
08677  6213  ce 32 11        dec xypos+1      Ord. high PIXEL CURS.
08678  6216  ce 31 11  ptfil2 dec vwork
08679  6219
08680  6219  a5 63           lda facexp       Flag für Stapeleintrag li.
08681  621b  20 7c 62        jsr tstval       Punkt testen
08682  621e  85 63           sta facexp       Flag für Stapeleintrag li.
08683  6220
08684  6220  18              clc
08685  6221  ad 31 11        lda vwork
08686  6224  69 02           adc #2
08687  6226  8d 31 11        sta vwork
08688  6229  90 03           bcc ptfil3
08689  622b  ee 32 11        inc xypos+1      Ord. high PIXEL CURS.
08690  622e
08691  622e  a5 64      ptfil3 lda facho      Flag für Stapeleintrag re.
08692  6230  20 7c 62        jsr tstval       Punkt testen
08693  6233  85 64           sta facho        Flag für Stapeleintrag re.
08694  6235
08695  6235  ae 31 11        ldx vwork        vorherige Abszisse
08696  6238  d0 03           bne ptfil4       wiederherstellen
08697  623a  ce 32 11        dec xypos+1      Ord. high PIXEL CURS.
08698  623d  ce 31 11  ptfil4 dec vwork
08699  6240
08700  6240  ee 33 11        inc xypos+2      Absz. low PIXEL CURS.
```

```
08701  6243  d0 03           bne ptfil6      nächsten Punkt testen, prüfen ob
                                             fertig
08702  6245  ee 34 11        inc xypos+3     Absz. High PIXEL CURS.
08703  6248
08704  6248
08705  6248  ===> nächsten Punkt testen, prüfen ob fertig <===
08706  6248
08707  6248  20 49 9c  ptfil6 jsr readpt     Test auf Punkt xr/yr gesetzt
08708  624b  b0 02           bcs ptmax       nicht im Graphikbereich
08709  624d  d0 bc           bne ptfill      gesetzt
08710  624f
08711  624f  a2 03     ptmax  ldx #3
08712  6251  a0 00           ldy #0
08713  6253
08714  6253  a5 25           lda index1+1    ADDRH Zeiger Koordinaten-Stapel
08715  6255  c5 34           cmp strend+1    ADDRH+1 Obergr. ARRAY Tabelle
08716  6257  d0 06           bne ptmax2      Stapel nicht leer
08717  6259
08718  6259  a5 24           lda index1      ADDRL Zeiger Koordinaten-Stapel
08719  625b  c5 33           cmp strend      ADDRL+1 Obergr. ARRAY Tabelle
08720  625d  f0 1a           beq ptdone      Stapel leer
08721  625f
08722  625f  a5 24     ptmax2 lda index1      ADDRL Zeiger Koordinaten-Stapel
08723  6261  d0 02           bne pain07
08724  6263  c6 25           dec index1+1    ADDRH Zeiger Koordinaten-Stapel
08725  6265  c6 24     pain07 dec index1      ADDRL Zeiger Koordinaten-Stapel
08726  6267
08727  6267  20 b7 03        jsr ind1_1      Koordinate aus Stapel nach ac
08728  626a  8d 03 ff        sta lcrc        ROM / PGM-RAM einschalten
08729  626d  9d 31 11        sta vwork,x     Stapel-Koord. in Koord. Tab.
08730  6270  ca              dex
08731  6271  10 ec           bpl ptmax2
08732  6273
08733  6273  20 b5 4b        jsr istpky      Stopp-Taste abfragen
08734  6276  4c eb 61        jmp ptloop      PAINT in x-Richtung ausführen
08735  6279
08736  6279
08737  6279  4c f2 9d  ptdone jmp dstpos     PAINT fertig
08738  627c
08739  627c
08740  627c  ===> Punkt testen <===
08741  627c
08742  627c  48        tstval pha            Flag Stapeleintrag
08743  627d
08744  627d  20 49 9c          jsr readpt     Test auf Punkt xr/yr gesetzt
08745  6280  b0 18           bcs tstgo       nicht im Graphikbereich
08746  6282  f0 16           beq tstgo       gesetzt
08747  6284
08748  6284  68              pla            Flag Stapeleintrag
08749  6285  d0 16           bne tstxit      schon gesetzt
08750  6287
08751  6287  aa              tax            0
08752  6288  a8              tay            0
08753  6289
08754  6289  a5 25           lda index1+1    ADDRH Zeiger Koordinaten-Stapel
08755  628b  c5 1c           cmp tempst+1    ADDRH max. Stapel-Bereich
08756  628d  90 0f           bcc ptsave
08757  628f  d0 06           bne pterr       Behandlung »?out of memory error«
```

```
08758  6291
08759  6291  a5 24          lda index1       ADDRL Zeiger Koordinaten-Stapel
08760  6293  c5 1b          cmp tempst       ADDRL max. Stapel-Bereich
08761  6295  90 07          bcc ptsave       nein
08762  6297
08763  6297  ===> Behandlung »?out of memory error« <===
08764  6297
08765  6297  4c 3a 4d  pterr  jmp omerr       Behandlung »?out of memory error«
08766  629a
08767  629a
08768  629a  68        tstgo  pla             Flag Stapeleintrag verwerfen
08769  629b  a9 00            lda #0          Flag kein Stapeleintrag
08770  629d  60        tstxit rts
08771  629e
08772  629e
08773  629e  bd 31 11  ptsave lda vwork,x
08774  62a1  8d 04 ff         sta lcrd        ROM / VAR-RAM einschalten
08775  62a4  91 24             sta (index1),y Koordinaten auf Stapel
08776  62a6  8d 03 ff         sta lcrc        ROM / PGM-RAM einschalten
08777  62a9
08778  62a9  e6 24             inc index1      ADDRL Zeiger Koordinaten-Stapel
08779  62ab  d0 02             bne paits2
08780  62ad  e6 25             inc index1+1    ADDRH Zeiger Koordinaten-Stapel
08781  62af
08782  62af  e8        paits2 inx
08783  62b0  e0 04            cpx #4
08784  62b2  d0 ea            bne ptsave
08785  62b4
08786  62b4  a9 80            lda #$80        Flag Stapeleintrag
08787  62b6  60               rts
08788  62b7
08789  62b7
08790  62b7  ===> BASIC-Statement BOX ($e1) <===
08791  62b7
08792  62b7  20 2f 9e  box    jsr grpcol      Test GRAPHIC, Farbqu. nach colsel
08793  62ba
08794  62ba  a2 1f            ldx #31         Koord. li. obere Ecke
08795  62bc  20 6d 9e         jsr incor3      Kommatest, Start-/Zielkoord. einl.
08796  62bf
08797  62bf  a2 2b            ldx #43         Koord. re. untere Ecke
08798  62c1  20 52 9e         jsr in2cor      Ersatz-/Start-/Zielkoord. einlesen
08799  62c4
08800  62c4  ===> Drehwinkel einlesen <===
08801  62c4
08802  62c4  20 06 9e         jsr optwrd      opt. 16 Bit INT nach yr/ac, sonst 0
08803  62c7  8c 54 11         sty boxang      Low-Byte Winkel
08804  62ca  8d 55 11         sta boxang+1    High-Byte Winkel
08805  62cd
08806  62cd  ===> Ausmal - Parameter einlesen <===
08807  62cd
08808  62cd  20 1c 9e         jsr optzer      Parameter Ausmalen
08809  62d0  e0 02            cpx #2
08810  62d2  90 03            bcc box05
08811  62d4
08812  62d4  ===> Behandlung »?illegal quantity error« <===
08813  62d4
08814  62d4  4c 28 7d         jmp fcerr       Behandlung »?illegal quantity error«
08815  62d7
```

```
08816  62d7
08817  62d7  8e 6c 11   box05  stx filflg       Flag 1=Ausmalen, 0=nicht
08818  62da
08819  62da  8a                txa
08820  62db  48                pha              Flag Ausmalen
08821  62dc
08822  62dc  20 89 63          jsr boxsub       Drehung ausführen
08823  62df
08824  62df  68                pla              Flag Ausmalen
08825  62e0  d0 1c             bne boxfil       Rechteck Fläche füllen
08826  62e2  f0 03             beq box15        Rechteck Grenzen zeichnen
08827  62e4
08828  62e4
08829  62e4  20 0b 64   box10  jsr boxpnt       Punkt gedrehtes Rechteck berechnen
08830  62e7
08831  62e7
08832  62e7  ===> Rechteck Grenzen zeichnen <===
08833  62e7
08834  62e7  20 30 9b   box15  jsr drawln       Strecke zeichnen
08835  62ea
08836  62ea  ad 4e 11          lda angcnt
08837  62ed  d0 f5             bne box10        noch nicht alle Grenzen
08838  62ef
08839  62ef  a2 04      box20  ldx #4
08840  62f1  bd 5b 11   box30  lda xcord2-1,x   Ausgangskoordinaten
08841  62f4  9d 30 11          sta vwork-1,x    wiederherstellen
08842  62f7  ca                dex
08843  62f8  d0 f7             bne box30
08844  62fa
08845  62fa  8e 6c 11          stx filflg       Flag 1=Ausmalen, 0=nicht
08846  62fd  60                rts
08847  62fe
08848  62fe
08849  62fe  ===> Rechteck Fläche füllen <===
08850  62fe
08851  62fe  a2 00      boxfil ldx #0
08852  6300
08853  6300  ad 49 11          lda angsgn       Quadrant
08854  6303  4a                lsr a
08855  6304  90 02             bcc boxf05       geradzahlig
08856  6306
08857  6306  a2 02             ldx #2
08858  6308  bd 60 11   boxf05 lda xrcos,x      Differenz Koordinaten
08859  630b  8d 5a 11          sta ydist2
08860  630e
08861  630e  bd 61 11          lda bitidx,x     Bitoffset
08862  6311  8d 5b 11          sta ysize
08863  6314
08864  6314  a9 00             lda #0
08865  6316  a2 03             ldx #3
08866  6318  9d 56 11   boxf10 sta xcount,x     Speicherbereich rücksetzen
08867  631b  ca                dex
08868  631c  10 fa             bpl boxf10
08869  631e
08870  631e  a2 07      boxf20 ldx #7
08871  6320  bd 31 11   boxf25 lda vwork,x
08872  6323  48                pha
08873  6324  ca                dex
```

```
08874  6325  10 f9           bpl boxf25
08875  6327
08876  6327  20 30 9b        jsr drawln       Strecke zeichnen
08877  632a
08878  632a  a2 00           ldx #0
08879  632c  68        boxf28 pla              Koordinaten
08880  632d  9d 31 11        sta vwork,x      wiederherstellen
08881  6330  e8              inx
08882  6331  e0 08           cpx #8
08883  6333  d0 f7           bne boxf28
08884  6335
08885  6335  ad 5a 11  boxf30 lda ydist2       Koordinatendifferenz
08886  6338  d0 05           bne boxf40       dekrementieren
08887  633a  ce 5b 11        dec ysize
08888  633d  30 b0           bmi box20        negativ, fertig
08889  633f
08890  633f  ce 5a 11  boxf40 dec ydist2
08891  6342
08892  6342  a2 25           ldx #37
08893  6344  a0 1b           ldy #cosval-vwork      Offset COS
08894  6346
08895  6346  ad 49 11        lda angsgn       Quadrant
08896  6349  4a              lsr a
08897  634a  90 02           bcc boxf45       geradzahlig
08898  634c
08899  634c  a0 19           ldy #sinval-vwork      Offset SIN
08900  634e
08901  634e  a9 00     boxf45 lda #%00000000   Flags Koord.erhöhung einr.
08902  6350  4a        boxf50 lsr a            Flags Koord.erhöhung
08903  6351
08904  6351  48              pha
08905  6352  20 6d 9d        jsr addtwo       Koord.(yr) plus Koord.(xr) nach
                                              ac/yr
08906  6355  9d 31 11        sta vwork,x
08907  6358  98              tya
08908  6359  9d 32 11        sta xypos+1,x    Ord. High PIXEL CURS.
08909  635c  68              pla
08910  635d
08911  635d  90 02           bcc boxf60
08912  635f
08913  635f  09 a0           ora #%10100000   Bit-Flags Koord.erhöhung
08914  6361
08915  6361  e8        boxf60 inx
08916  6362  e8              inx
08917  6363
08918  6363  a0 19           ldy #sinval-vwork      Offset SIN
08919  6365  4e 49 11        lsr angsgn       Quadrant
08920  6368  90 02           bcc boxf65       geradzahlig
08921  636a
08922  636a  a0 1b           ldy #cosval-vwork      Offset COS
08923  636c
08924  636c  2e 49 11  boxf65 rol angsgn       Quadrant
08925  636f
08926  636f  e0 27           cpx #ycount-vwork
08927  6371  f0 dd           beq boxf50       zweite Koordinate auch
08928  6373
08929  6373  a2 06           ldx #6
08930  6375  0a              asl a            Bit-Flags Koord.erhöhung
```

```
08931  6376  f0 bd          beq boxf30
08932  6378
08933  6378  90 08    boxf70 bcc boxf80       Bit nicht gesetzt
08934  637a
08935  637a  fe 31 11        inc vwork,x
08936  637d  d0 03           bne boxf80
08937  637f  fe 32 11        inc xypos+1,x     Ord. High PIXEL CURS.
08938  6382
08939  6382  0a        boxf80 asl a            nächstes Bit
08940  6383
08941  6383  ca             dex
08942  6384  ca             dex
08943  6385  10 f1          bpl boxf70
08944  6387  30 95          bmi boxf20
08945  6389
08946  6389
08947  6389  ===> Drehung ausführen <===
08948  6389
08949  6389  a0 23    boxsub ldy #boxang-vwork      Offset Drehwinkel
08950  638b  20 74 9a        jsr getand        SIN/COS Winkel (yr) nach sinpc/cospc
08951  638e
08952  638e  a2 1f           ldx #xcord1-vwork
08953  6390  a0 2b           ldy #xcord2-vwork
08954  6392
08955  6392  98        boxsb1 tya
08956  6393  48             pha
08957  6394
08958  6394  20 99 9d        jsr abstwo        Abs.wert Koord.Diff. (yr,xr) nach
                                                ac, yr
08959  6397  9d 35 11        sta xdest,x
08960  639a  9d 39 11        sta xyabs,x
08961  639d  9d 41 11        sta fct,x
08962  63a0
08963  63a0  98              tya
08964  63a1  9d 36 11        sta xdest+1,x
08965  63a4  9d 3a 11        sta xyabs+1,x
08966  63a7  9d 42 11        sta fct+1,x
08967  63aa
08968  63aa  68              pla
08969  63ab  a8              tay
08970  63ac  20 6d 9d        jsr addtwo        Koord.(yr) plus Koord.(xr) nach
                                                ac/yr
08971  63af
08972  63af  9d 31 11        sta vwork,x
08973  63b2
08974  63b2  98              tya
08975  63b3  9d 32 11        sta vwork+1,x
08976  63b6
08977  63b6  a0 2d           ldy #ycord2-vwork
08978  63b8
08979  63b8  e8              inx
08980  63b9  e8              inx
08981  63ba  e0 21           cpx #ycord1-vwork
08982  63bc  f0 d4           beq boxsb1
08983  63be
08984  63be  a9 90           lda #%10010000    SIN (0) bzw. COS (1) Flags
08985  63c0  20 f3 9a        jsr angdst        Dist.Koord. mit SIN/COS mult.
08986  63c3
```

```
08987  63c3  ad 49 11          lda angsgn       Quadrant
08988  63c6  29 03             and #%00000011
08989  63c8  8d 49 11          sta angsgn       Quadrant
08990  63cb
08991  63cb  aa                tax
08992  63cc  bd ed 63          lda boxtab,x     Vorzeichenstring
08993  63cf  20 0b 64          jsr boxpnt       Punkt gedrehtes Rechteck berechnen
08994  63d2  20 f2 9d          jsr dstpos       Zielkoord. werden akt. Koord.
08995  63d5
08996  63d5  ad 4e 11          lda angcnt       plus-minus-Flags
08997  63d8  20 0b 64          jsr boxpnt       Punkt gedrehtes Rechteck berechnen
08998  63db
08999  63db  ae 49 11          ldx angsgn       Quadrant
09000  63de
09001  63de  bd ed 63          lda boxtab,x     Vorzeichenstring
09002  63e1  29 f0             and #%11110000
09003  63e3  8d 4f 11          sta angcnt+1     Vorz.-Flag
09004  63e6
09005  63e6  bd f1 63          lda angtab,x
09006  63e9  8d 4e 11          sta angcnt
09007  63ec  60                rts
09008  63ed
09009  63ed
09010  63ed  be e4 41   boxtab .byte %10111110, %11100100, %01000001, %00011011
09010  63f0  1b
09011  63f1
09012  63f1  41 1b be   angtab .byte %01000001, %00011011, %10111110, %11100100
09012  63f4  e4
09013  63f5
09014  63f5
09015  63f5  ===> Namen der Programmautoren <===
09016  63f5
09017  63f5  46 52 45          .byte 'fred b', _cr    Fred Bowen
09017  63f8  44 20 42
09017  63fb  0d
09018  63fc  54 45 52          .byte 'terry r', _cr   Terry Ryan
09018  63ff  52 59 20
09018  6402  52 0d
09019  6404  4d 49 4b          .byte 'mike i', _cr
09019  6407  45 20 49
09019  640a  0d
09020  640b
09021  640b  erstellten bis 06/1985 bei Commodore Westchester (USA)
09022  640b  die C 128-Software mit Cross-Assembler auf einer Sun.
09023  640b
09024  640b
09025  640b  ===> Punkt gedrehtes Rechteck berechnen <===
09026  640b
09027  640b  20 67 67   boxpnt jsr anglpt       Berechnung der doppelten Koordinaten
09028  640e
09029  640e  a2 04             ldx #4
09030  6410  bd 32 11   boxp10 lda vwork+1,x
09031  6413  0a                asl a
09032  6414
09033  6414  7e 32 11          ror vwork+1,x    Koordinaten durch
09034  6417  7e 31 11          ror vwork,x      2 dividieren
09035  641a  90 08             bcc boxp20
09036  641c
```

```
09037  641c  fe 31 11          inc vwork,x
09038  641f  d0 03             bne boxp20        erhöhen
09039  6421  fe 32 11          inc vwork+1,x
09040  6424
09041  6424  e8          boxp20 inx
09042  6425  e8                 inx
09043  6426
09044  6426  e0 06             cpx #6
09045  6428  f0 e6             beq boxp10
09046  642a
09047  642a  60                 rts
09048  642b
09049  642b
09050  642b  ===> BASIC-Statement SSHAPE ($e4) <===
09051  642b
09052  642b  20 74 a0   sshape jsr isgrap       Test GRAPHIC-RAM reserviert
09053  642e  20 af 7a          jsr ptrget       folg. Variable suchen/anlegen
09054  6431
09055  6431  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
09056  6434
09057  6434  8d 5f 11          sta rowcnt       ADDRL Zeiger auf Descriptor
09058  6437  8c 60 11          sty xrcos        ADDRH Zeiger auf Descriptor
09059  643a
09060  643a  24 0f             bit valtyp       Flag Variablentyp
09061  643c  30 03             bmi havstr       String
09062  643e
09063  643e  ===> Behandlung »?type mismatch error« <===
09064  643e
09065  643e  4c e7 77          jmp chkerr       Behandlung »?type mismatch error«
09066  6441
09067  6441
09068  6441  a2 28      havstr ldx #xsize-vwork Start Koord.
09069  6443  20 6d 9e          jsr incor3       Kommatest, Start-/Zielkoord. einl.
09070  6446
09071  6446  a2 04             ldx #xdest-vwork Ziel Koord.
09072  6448  20 52 9e          jsr in2cor       Ersatz-/Start-/Zielkoord. einlesen
09073  644b
09074  644b  a2 2a             ldx #ysize-vwork
09075  644d  a0 06             ldy #ydest-vwork
09076  644f
09077  644f  a9 02             lda #2
09078  6451  85 8e             sta vtemp1
09079  6453
09080  6453  20 99 9d   boxs05 jsr abstwo       Abs.wert Koord.Diff. (yr,xr) nach
                                                ac, yr
09081  6456
09082  6456  aa                tax              Low-Byte Abs.wert
09083  6457  98                tya              High-Byte Abs.wert
09084  6458  48                pha
09085  6459
09086  6459  a4 8e             ldy vtemp1
09087  645b  20 f9 9d          jsr dstmov       eine Zielkoord. nach akt. Koord.
09088  645e  90 0c             bcc boxs10       Koord.differenz war negativ
09089  6460
09090  6460  b9 59 11          lda xsize,y
09091  6463  99 31 11          sta vwork,y
09092  6466
09093  6466  b9 5a 11          lda xsize+1,y
```

```
09094  6469  99 32 11          sta vwork+1,y
09095  646c
09096  646c  8a          boxs10 txa
09097  646d  99 59 11          sta xsize,y
09098  6470  99 db 03          sta savsiz,y     Low-Byte Breite SHAPE Bereich
09099  6473
09100  6473  68                pla
09101  6474  99 5a 11          sta xsize+1,y
09102  6477  99 dc 03          sta savsiz+1,y   High-Byte Breite SHAPE Bereich
09103  647a
09104  647a  a2 28             ldx #xsize-vwork
09105  647c  a0 04             ldy #xdest-vwork
09106  647e
09107  647e  c6 8e             dec vtemp1
09108  6480  c6 8e             dec vtemp1
09109  6482  f0 cf             beq boxs05       zweimal
09110  6484
09111  6484  a0 ff             ldy #$ff
09112  6486  8c 55 11          sty strptr       Stringzeiger
09113  6489
09114  6489  ad 31 11          lda xypos        Ord. Low PIXEL CURS.
09115  648c  8d 5d 11          sta xsave
09116  648f
09117  648f  ad 32 11          lda xypos+1      Ord. High PIXEL CURS.
09118  6492  8d 5e 11          sta xsave+1
09119  6495
09120  6495  98                tya              =$ff
09121  6496  20 90 86          jsr strspa       Arb.string Länge ac anlegen
09122  6499
09123  6499  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
09124  649c
09125  649c  20 e3 9c  boxs15 jsr getpos       Graph.pnt., Pix.Curs., Bitmaske
                                                berechnen
09126  649f  b1 8c             lda (grapnt),y   Byte aus GRAPHIC RAM
09127  64a1  90 0e             bcc boxs25       im Graphikbereich
09128  64a3
09129  64a3  ad 31 11          lda vwork
09130  64a6
09131  64a6  24 d8             bit graphm       Flag Text/Graphik/Split-BS
09132  64a8  10 02             bpl boxs20       nicht MULTICOLOR
09133  64aa
09134  64aa  38                sec
09135  64ab  2a                rol a
09136  64ac
09137  64ac  29 07       boxs20 and #%00000111   Bit Offset
09138  64ae  aa                tax
09139  64af
09140  64af  a9 00             lda #0
09141  64b1  24 d8       boxs25 bit graphm       Flag Text/Graphik/Split-BS
09142  64b3  10 01             bpl boxs30       nein
09143  64b5
09144  64b5  ca                dex              Bit Offset
09145  64b6  8e 61 11    boxs30 stx bitidx       Bitoffset
09146  64b9
09147  64b9  0a          boxs35 asl a
09148  64ba  ca                dex
09149  64bb  10 fc             bpl boxs35
09150  64bd
```

```
09151  64bd  6a              ror a
09152  64be  85 8e           sta vtemp1
09153  64c0
09154  64c0  a9 08           lda #8
09155  64c2
09156  64c2  24 d8           bit graphm       Flag Text/Graphik/Split-BS
09157  64c4  10 01           bpl boxs40       nicht MULTICOLOR
09158  64c6
09159  64c6  4a              lsr a
09160  64c7  18       boxs40 clc
09161  64c8  6d 31 11        adc vwork
09162  64cb  8d 31 11        sta vwork
09163  64ce
09164  64ce  90 03           bcc boxs45
09165  64d0  ee 32 11        inc xypos+1      Ord. High PIXEL CURS.
09166  64d3
09167  64d3  20 e3 9c boxs45 jsr getpos       Graph.pnt., Pix.Curs., Bitmaske
                                              berechnen
09168  64d6  a9 00           lda #0
09169  64d8  b0 02           bcs boxs50       nicht im Graphikbereich
09170  64da
09171  64da  b1 8c           lda (grapnt),y   Byte aus Bitspeicher
09172  64dc  85 8f    boxs50 sta vtemp2
09173  64de
09174  64de  ae 61 11        ldx bitidx       Bitoffset
09175  64e1  4a       boxs55 lsr a
09176  64e2  e8              inx
09177  64e3  e0 08           cpx #8
09178  64e5  d0 fa           bne boxs55
09179  64e7
09180  64e7  05 8e           ora vtemp1
09181  64e9  ee 55 11        inc strptr       Stringzeiger
09182  64ec
09183  64ec  ac 55 11        ldy strptr       Stringzeiger
09184  64ef  c0 fc           cpy #$fc         Test auf Platz in String
09185  64f1
09186  64f1  letzte 4 Bytes für Breite/Höhe SHAPE Bereich
09187  64f1
09188  64f1  90 03           bcc boxs60       ja
09189  64f3
09190  64f3  4c ed a5        jmp errlen       Behandlung »?string too long«
09191  64f6
09192  64f6
09193  64f6  8d 04 ff boxs60 sta lcrd         ROM / VAR-RAM einschalten
09194  64f9  91 64           sta (facho),y    Byte in String
09195  64fb  8d 03 ff        sta lcrc         ROM / PGM-RAM einschalten
09196  64fe
09197  64fe  ae 61 11        ldx bitidx       Bitoffset
09198  6501  ad 59 11        lda xsize
09199  6504
09200  6504  38              sec
09201  6505  24 d8           bit graphm       Flag Text/Graphik/Split-BS
09202  6507  10 03           bpl boxs65       nein
09203  6509
09204  6509  e9 04           sbc #4
09205  650b  2c              .byte _skip2
09206  650c
09207  650c  e9 08    boxs65 sbc #8
```

```
09208  650e  8d 59 11          sta xsize
09209  6511
09210  6511  a5 8f             lda vtemp2       letztes Byte aus Bitspeicher
09211  6513  b0 a4             bcs boxs35
09212  6515
09213  6515  ce 5a 11          dec ydist2
09214  6518  10 9f             bpl boxs35
09215  651a
09216  651a  ae 5b 11          ldx ysize
09217  651d  d0 48             bne boxs80
09218  651f
09219  651f  ce 5c 11          dec xcord2
09220  6522  10 43             bpl boxs80
09221  6524
09222  6524  24 d8             bit graphm       Flag Text/Graphik/Split-BS
09223  6526  10 06             bpl boxs70       nicht MULTICOLOR
09224  6528
09225  6528  0e db 03          asl savsiz       Low-Byte Breite SHAPE Bereich
09226  652b  2e dc 03          rol savsiz+1     High-Byte Breite SHAPE Bereich
09227  652e
09228  652e  a2 00       boxs70 ldx #0
09229  6530  bd db 03    boxs75 lda savsiz,x     Breite und Höhe als letzte 4 Bytes
09230  6533  c8                iny
09231  6534  8d 04 ff          sta lcrd         ROM / VAR-RAM einschalten
09232  6537  91 64             sta (facho),y    FAC: M4
09233  6539  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
09234  653c
09235  653c  e8                inx
09236  653d  e0 04             cpx #4
09237  653f  d0 ef             bne boxs75
09238  6541
09239  6541  ===> Stringdescriptor SHAPE String <===
09240  6541
09241  6541  c8                iny
09242  6542  8c db 03          sty savsiz       Stringlänge
09243  6545
09244  6545  a5 64             lda facho        ADDRL Stringadresse
09245  6547  8d dc 03          sta savsiz+1
09246  654a
09247  654a  a5 65             lda facmoh       ADDRH Stringadresse
09248  654c  8d dd 03          sta savsiz+2
09249  654f
09250  654f  a9 db             lda #<savsiz     ADDRL Descriptor Arb.string
09251  6551  85 66             sta indice
09252  6553
09253  6553  a9 03             lda #>savsiz     ADDRH Descriptor Arb.string
09254  6555  85 67             sta indice+1
09255  6557
09256  6557  ad 5f 11          lda rowcnt       ADDRL Zeiger auf Descriptor
09257  655a  85 4b             sta lstpnt
09258  655c
09259  655c  ad 60 11          lda xrcos        ADDRH Zeiger auf Descriptor
09260  655f  85 4c             sta lstpnt+1
09261  6561
09262  6561  20 94 54          jsr copy         String ins RAM, Test/Trailer
09263  6564  4c f2 9d          jmp dstpos       Zielkoord. werden akt. Koord.
09264  6567
09265  6567
```

```
09266  6567  ce 5b 11   boxs80 dec ysize
09267  656a
09268  656a  ee 33 11          inc xypos+2       Absz. Low PIXEL CURS.
09269  656d  d0 03             bne boxs85
09270  656f  ee 34 11          inc xypos+3       Absz. High PIXEL CURS.
09271  6572
09272  6572  ad 5d 11   boxs85 lda xsave
09273  6575  8d 31 11          sta vwork
09274  6578
09275  6578  ad 5e 11          lda ycord2
09276  657b  8d 32 11          sta xypos+1       Ord. High PIXEL CURS.
09277  657e
09278  657e  ad db 03          lda savsiz        Low-Byte Breite SHAPE Bereich
09279  6581  8d 59 11          sta xsize
09280  6584
09281  6584  ad dc 03          lda savsiz+1      High-Byte Breite SHAPE Bereich
09282  6587  8d 5a 11          sta ydist2
09283  658a
09284  658a  4c 9c 64          jmp boxs15
09285  658d
09286  658d
09287  658d                    .end
09288  658d                    .lib basic.graph_2
09289  658d  put"@basic.graph_2"
```

```
09291  658d
09292  658d
09293  658d  ===> BASIC-Statement GSHAPE ($e3) <===
09294  658d
09295  658d  20 74 a0   gshape jsr isgrap       Test GRAPHIC-RAM reserviert
09296  6590  20 7b 87          jsr sav13        Ausdruck in Arbeitsstring, frestr
09297  6593
09298  6593  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
09299  6596
09300  6596  8d 53 11          sta strsz        Stringlänge
09301  6599  86 26             stx index2       ADDRL String
09302  659b  84 27             sty index2+1     ADDRH String
09303  659d
09304  659d  ===> Koord. linke ob. Ecke einlesen <===
09305  659d
09306  659d  a2 04             ldx #4
09307  659f  20 52 9e          jsr in2cor       Ersatz-/Start-/Zielkoord. einlesen
09308  65a2
09309  65a2  ===> Modus (0-4) einlesen <===
09310  65a2
09311  65a2  20 1c 9e          jsr optzer       opt. 8 Bit INT nach xr, sonst 0
09312  65a5  e0 05             cpx #5
09313  65a7  90 03             bcc gsh010
09314  65a9
09315  65a9  ===> Behandlung »?illegal quantity error« <===
09316  65a9
09317  65a9  4c 28 7d          jmp fcerr        Behandlung »?illegal quantity error«
09318  65ac
09319  65ac
09320  65ac  8e 54 11   gsh010 stx gettyp       Modus GSHAPE
09321  65af  a2 03             ldx #3
09322  65b1
09323  65b1  ac 53 11          ldy strsz        Stringlänge
09324  65b4  c0 05             cpy #5
09325  65b6  b0 01             bcs gsh020       Breite / Höhe SHAPE aus String
09326  65b8
09327  65b8  nur 4 Bytes für Breite / Höhe oder Nullstring
09328  65b8
09329  65b8  60                rts
09330  65b9
09331  65b9
09332  65b9  ===> Breite / Höhe SHAPE aus String <===
09333  65b9
09334  65b9  88         gsh020 dey
09335  65ba  a9 26             lda #<index2     ADDRL Stringadresse
09336  65bc  20 ab 03          jsr iram1        LDA (ac),y aus VAR-RAM
09337  65bf
09338  65bf  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
09339  65c2
09340  65c2  9d 59 11          sta xsize,x
09341  65c5  ca                dex
09342  65c6  10 f1             bpl gsh020       Breite / Höhe SHAPE aus String
09343  65c8
09344  65c8  8e 55 11          stx strptr       Stringzeiger einrichten $ff
09345  65cb  20 f2 9d          jsr dstpos       Zielkoord. werden akt. Koord.
09346  65ce
09347  65ce  ad 59 11          lda xsize        Low-Byte SHAPE Breite
09348  65d1  8d 5d 11          sta xsave        Low-Byte SHAPE Breite
```

```
09349  65d4
09350  65d4  ad 5a 11          lda xsize+1      High-Byte SHAPE Breite
09351  65d7  8d 5e 11          sta xsave+1      High-Byte SHAPE Breite
09352  65da
09353  65da  a9 08      gsh030 lda #8
09354  65dc  8d 69 11          sta bitcnt       Bitzähler
09355  65df  ee 55 11          inc strptr       Stringzeiger
09356  65e2
09357  65e2  ac 55 11          ldy strptr       Stringzeiger
09358  65e5  a9 26             lda #<index2     Zeiger auf SHAPE String
09359  65e7  20 ab 03          jsr iram1        LDA (ac),y aus VAR-RAM
09360  65ea
09361  65ea  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
09362  65ed
09363  65ed  8d 57 11          sta newbyt       Byte aus SHAPE String
09364  65f0
09365  65f0  20 49 9c  gsh040  jsr readpt       Test auf Punkt xr/yr gesetzt
09366  65f3  8d 56 11          sta xcount
09367  65f6
09368  65f6  0e 57 11          asl newbyt       Byte aus SHAPE String
09369  65f9  2a                rol a
09370  65fa  ce 69 11          dec bitcnt       Bitzähler
09371  65fd
09372  65fd  24 d8             bit graphm       Flag Text/Graphik/Split-BS
09373  65ff  10 07             bpl gsh050       nicht MULTICOLOR
09374  6601
09375  6601  0e 57 11          asl newbyt       Byte aus SHAPE String
09376  6604  2a                rol a
09377  6605  ce 69 11          dec bitcnt       Bitzähler
09378  6608
09379  6608  ae 54 11  gsh050  ldx gettyp       Modus GSHAPE
09380  660b  e0 03             cpx #3
09381  660d  90 0c             bcc gsh070
09382  660f  f0 05             beq gsh060       GSHAPE Modus 3 (log. UND)
09383  6611
09384  6611  ===> GSHAPE Modus 4 (löschen) <===
09385  6611
09386  6611  4d 56 11          eor xcount
09387  6614  b0 11             bcs gsh090       immer
09388  6616
09389  6616
09390  6616  ===> GSHAPE Modus 3 (log. UND) <===
09391  6616
09392  6616  2d 56 11  gsh060  and xcount
09393  6619  b0 0c             bcs gsh090       immer
09394  661b
09395  661b
09396  661b  e0 01      gsh070  cpx #1
09397  661d  90 08             bcc gsh090       GSHAPE Modus 0 (orig.)
09398  661f  f0 04             beq gsh080       GSHAPE Modus 1 (invertieren)
09399  6621
09400  6621  ===> GSHAPE Modus 2 (log. ODER) <===
09401  6621
09402  6621  0d 56 11          ora xcount       = 2, OR-Modus
09403  6624  2c                .byte _skip2
09404  6625
09405  6625
09406  6625
```

```
09406  6625  ===> GSHAPE Modus 1 (invertieren) <===
09407  6625
09408  6625  49 ff      gsh080 eor #%11111111
09409  6627
09410  6627  29 03      gsh090 and #%00000011
09411  6629
09412  6629  24 d8             bit graphm       Flag Text/Graphik/Split-BS
09413  662b  30 02             bmi gsh100       MULTICOLOR
09414  662d
09415  662d  29 01             and #%00000001
09416  662f
09417  662f  85 83      gsh100 sta colsel       Farbquelle (0-3)
09418  6631  20 19 9c          jsr plot01       Graphik Punkt setzen
09419  6634
09420  6634  ee 31 11          inc vwork
09421  6637  d0 03             bne gsh110
09422  6639  ee 32 11          inc xypos+1      Ord. High PIXEL CURS.
09423  663c
09424  663c  38        gsh110 sec
09425  663d  ad 5d 11          lda xsave        Low-Byte SHAPE Breite
09426  6640
09427  6640  24 d8             bit graphm       Flag Text/Graphik/Split-BS
09428  6642  10 03             bpl gsh120       nicht MULTICOLOR
09429  6644
09430  6644  e9 02             sbc #2
09431  6646  2c                .byte _skip2
09432  6647
09433  6647  e9 01      gsh120 sbc #1
09434  6649  8d 5d 11          sta xsave        Low-Byte SHAPE Breite
09435  664c
09436  664c  ad 5e 11          lda ycord2       High-Byte SHAPE Breite
09437  664f  e9 00             sbc #0
09438  6651  8d 5e 11          sta ycord2       High-Byte SHAPE Breite
09439  6654  b0 2d             bcs gsh150
09440  6656
09441  6656  ===> PIXEL CURS. linke Ecke SHAPE <===
09442  6656
09443  6656  a2 01             ldx #1
09444  6658  bd 59 11  gsh130 lda xsize,x
09445  665b  9d 5d 11          sta xsave,x
09446  665e
09447  665e  bd 35 11          lda xdest,x
09448  6661  9d 31 11          sta vwork,x
09449  6664
09450  6664  ca                dex
09451  6665  10 f1             bpl gsh130
09452  6667
09453  6667  ee 33 11          inc xypos+2      Absz. Low PIXEL CURS.
09454  666a  d0 03             bne gsh140
09455  666c  ee 34 11          inc xypos+3      Absz. High PIXEL CURS.
09456  666f
09457  666f  38        gsh140 sec
09458  6670  ad 5b 11          lda ysize        Low-Byte SHAPE Höhe
09459  6673  e9 01             sbc #1
09460  6675  8d 5b 11          sta ysize        Low-Byte SHAPE Höhe
09461  6678
09462  6678  ad 5c 11          lda ysize+1      High-Byte SHAPE Höhe
09463  667b  e9 00             sbc #0
```

```
09464  667d  8d 5c 11          sta ysize+1      High-Byte SHAPE Höhe
09465  6680  b0 09             bcs gsh160
09466  6682
09467  6682  60                rts
09468  6683
09469  6683
09470  6683  ad 69 11   gsh150 lda bitcnt       Bitzähler
09471  6686  f0 03             beq gsh160
09472  6688
09473  6688  4c f0 65          jmp gsh040
09474  668b
09475  668b
09476  668b  4c da 65   gsh160 jmp gsh030
09477  668e
09478  668e
09479  668e  ===> BASIC-Statement CIRCLE ($e2) <===
09480  668e
09481  668e  20 2f 9e   circle jsr grpcol       Test GRAPHIC, Farbqu. nach colsel
09482  6691
09483  6691  ===> Koord. Mittelpunkt einlesen <===
09484  6691
09485  6691  a2 1f             ldx #xcircl-vwork
09486  6693  20 52 9e          jsr in2cor       Ersatz-/Start-/Zielkoord. einlesen
09487  6696
09488  6696  ===> x-Radius einlesen <===
09489  6696
09490  6696  20 06 9e          jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
09491  6699  8c 54 11          sty xradus       Low-Byte x-Radius
09492  669c  8d 55 11          sta xradus+1     High-Byte x-Radius
09493  669f
09494  669f  ===> y-Radius einlesen <===
09495  669f
09496  669f  20 06 9e          jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
09497  66a2  8c 56 11          sty yradus       Low-Byte y-Radius
09498  66a5  8d 57 11          sta yradus+1     High-Byte y-Radius
09499  66a8  08                php
09500  66a9
09501  66a9  ===> SCALE <===
09502  66a9
09503  66a9  a2 23             ldx #xradus-vwork
09504  66ab  20 4a 9d          jsr scalxy       Test/SCALE: Koord. umrechnen
09505  66ae
09506  66ae  28                plp
09507  66af  b0 11             bcs circ05
09508  66b1
09509  66b1  ===> Vorgabe für y-Radius (=x-Radius) <===
09510  66b1
09511  66b1  ad 54 11          lda xradus       Low-Byte x-Radius
09512  66b4  8d 56 11          sta yradus       Low-Byte y-Radius
09513  66b7
09514  66b7  ad 55 11          lda xradus+1     High-Byte x-Radius
09515  66ba
09516  66ba  24 d8             bit graphm       Flag Text/Graphik/Split-BS
09517  66bc  10 04             bpl circ05       nicht MULTICOLOR
09518  66be
09519  66be  0e 56 11          asl yradus       Low-Byte y-Radius
09520  66c1  2a                rol a            High-Byte y-Radius
09521  66c2
```

```
09522  66c2  8d 57 11   circ05 sta yradus+1     High-Byte y-Radius
09523  66c5
09524  66c5  ===> Startwinkel Kreisbogen einlesen <===
09525  66c5
09526  66c5  20 06 9e          jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
09527  66c8  8c 5c 11          sty xcord2       Low-Byte Startwinkel
09528  66cb  8d 5d 11          sta xcord2+1     High-Byte Startwinkel
09529  66ce
09530  66ce  ===> Endwinkel Kreisbogen einlesen <===
09531  66ce
09532  66ce  20 06 9e          jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
09533  66d1  8c 5e 11          sty ycord2       Low-Byte Endwinkel
09534  66d4  8d 5f 11          sta ycord2+1     High-Byte Endwinkel
09535  66d7
09536  66d7  ===> Drehwinkel einlesen <===
09537  66d7
09538  66d7  20 06 9e          jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
09539  66da  85 77             sta mltus1
09540  66dc
09541  66dc  98                tya              YR und AC vertauschen
09542  66dd  a4 77             ldy mltus1
09543  66df  20 77 9a          jsr gtang1       SIN/COS ac/yr nach sinpc/cospc
09544  66e2
09545  66e2  a2 2d             ldx #ycord2-vwork      Koord. Endwinkel
09546  66e4  a0 2b             ldy #xcord2-vwork      Koord. Startwinkel
09547  66e6  20 7c 9d          jsr subtwo       Koord.(yr) minus Koord.(xr) nach
                                                ac/yr
09548  66e9  90 0e             bcc circ10       Startwinkel < Endwinkel
09549  66eb
09550  66eb  ===> Endwinkel um 360 Grad erhöhen <===
09551  66eb
09552  66eb  a9 68             lda #<360
09553  66ed  a0 01             ldy #>360
09554  66ef  20 70 9d          jsr addtw2       ac/yr plus Koord.(xr) nach ac/yr
09555  66f2
09556  66f2  9d 31 11          sta vwork,x      Low-Byte Endwinkel
09557  66f5  98                tya
09558  66f6  9d 32 11          sta vwork+1,x    High-Byte Endwinkel
09559  66f9
09560  66f9  a2 03       circ10 ldx #3
09561  66fb
09562  66fb  bd 54 11    circ20 lda xradus,x
09563  66fe  9d 58 11           sta rotang,x
09564  6701  ca                 dex
09565  6702  10 f7              bpl circ20
09566  6704
09567  6704  a9 90              lda #%10010000   Bit Flags SIN (0) / COS (1)
09568  6706  20 f3 9a          jsr angdst       Dist.Koord. mit SIN/COS mult.
09569  6709
09570  6709  a2 07              ldx #7
09571  670b
09572  670b  bd 54 11    circ30 lda xradus,x     x-Radius
09573  670e  9d 60 11           sta xrcos,x
09574  6711  ca                 dex
09575  6712  10 f7              bpl circ30
09576  6714
09577  6714  20 50 67          jsr cirsub       Drehung ausführen
09578  6717  20 f2 9d          jsr dstpos       Zielkoord. werden akt. Koord.
```

```
09579  671a
09580  671a
09580  671a  ===> Segmentwinkel <256 Grad einlesen <===
09581  671a
09582  671a  a2 02           ldx #2          Vorgabe 2 Grad
09583  671c  20 1e 9e        jsr optbyt      opt. 8 Bit INT nach xr
09584  671f  8a              txa             Segmentwinkel
09585  6720  d0 03           bne circ40
09586  6722
09587  6722  ===> Behandlung »?illegal quantity error« <===
09588  6722
09589  6722  4c 28 7d        jmp fcerr       Behandlung »?illegal quantity error«
09590  6725
09591  6725
09592  6725  8e 20 12  circ40 stx crcseg      Segmentwinkel
09593  6728
09594  6728  18              clc
09595  6729
09596  6729
09597  6729  ===> nächstes Segment zeichnen <===
09598  6729
09599  6729  ad 20 12  circ45 lda crcseg      Segmentwinkel
09600  672c  6d 5c 11        adc xcord2      Low-Byte Startwinkel
09601  672f  8d 5c 11        sta xcord2      Low-Byte Startwinkel
09602  6732
09603  6732  90 03           bcc circ50
09604  6734  ee 5d 11        inc xcord2+1    High-Byte Startwinkel
09605  6737
09606  6737  a2 2d     circ50 ldx #ycord2-vwork     Koord. Endwinkel
09607  6739  a0 2b            ldy #xcord2-vwork     Koord. akt. Segment
09608  673b  20 7c 9d        jsr subtwo      Koord.(yr) minus Koord.(xr) nach
                                             ac/yr
09609  673e  b0 08           bcs cirend      Differenz > 0
09610  6740
09611  6740  20 50 67        jsr cirsub      Drehung ausführen
09612  6743  20 30 9b        jsr drawln      Strecke zeichnen
09613  6746  90 e1           bcc circ45      immer
09614  6748
09615  6748
09616  6748  a0 2d     cirend ldy #ycord2-vwork     Koord. Endwinkel
09617  674a  20 52 67        jsr cirs10
09618  674d  4c 30 9b        jmp drawln      Strecke zeichnen
09619  6750
09620  6750
09621  6750  ===> Drehung ausführen <===
09622  6750
09623  6750  a0 2b     cirsub ldy #xcord2-vwork     Koord. akt. Segment
09624  6752  20 74 9a  cirs10 jsr getand      SIN/COS Winkel (yr) nach sinpc/cospc
09625  6755
09626  6755  a2 07           ldx #7
09627  6757
09628  6757  bd 60 11  cirs20 lda xrcos,x
09629  675a  9d 54 11        sta xradus,x    x-Radius
09630  675d  ca              dex
09631  675e  10 f7           bpl cirs20
09632  6760
09633  6760  a9 50           lda #%01010000  Bit Flags SIN (0) / COS (1)
09634  6762  20 f3 9a        jsr angdst      Dist.Koord. mit SIN/COS mult.
```

```
09635  6765
09636  6765  a9 10            lda #%00010000   Bit Flags SIN (0) / COS (1)
09637  6767
09638  6767
09639  6767  ===> Berechnung der doppelten Koordinaten <===
09640  6767
09641  6767  8d 4e 11   anglpt sta angcnt
09642  676a
09643  676a  a0 1f            ldy #xcircl-vwork
09644  676c  a2 23            ldx #xradus-vwork
09645  676e
09646  676e  0e 4f 11   angl10 asl angcnt+1     Flag für +/-
09647  6771  2e 4e 11          rol angcnt
09648  6774
09649  6774  20 6b 9d          jsr dotwo        Koord.(yr) plus/minus Koord.(xr)
                                                nach ac/yr
09650  6777
09651  6777  e8               inx
09652  6778  e8               inx
09653  6779
09654  6779  0e 4f 11          asl angcnt+1     Flag für +/-
09655  677c  2e 4e 11          rol angcnt
09656  677f  20 67 9d          jsr dotwo2       ac/yr plus/minus Koord.(xr) nach
                                                ac/yr
09657  6782
09658  6782  48               pha              Low-Byte Resultat
09659  6783  98               tya              Stapel ablegen
09660  6784  48               pha              High-Byte Resultat
09661  6785
09662  6785  a0 21            ldy #ycircl-vwork
09663  6787
09664  6787  e8               inx
09665  6788  e8               inx
09666  6789
09667  6789  e0 27            cpx #xradus+4-vwork
09668  678b  f0 e1            beq angl10
09669  678d
09670  678d  a2 03            ldx #3
09671  678f  68         angl20 pla              Resultate vom Stapel
09672  6790  9d 35 11          sta xdest,x
09673  6793  ca               dex
09674  6794  10 f9            bpl angl20
09675  6796
09676  6796  60               rts
09677  6797
09678  6797
09679  6797  ===> BASIC-Statement DRAW ($e5) <===
09680  6797
09681  6797  20 74 a0   draw   jsr isgrap       Test GRAPHIC-RAM reserviert
09682  679a
09683  679a  a2 01            ldx #1           Vorgabe Vordergrund
09684  679c  86 83            stx colsel       Farbquelle (0-3)
09685  679e
09686  679e  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
09687  67a1  c9 a4            cmp #_tot        TO Token
09688  67a3  f0 0b            beq draw10
09689  67a5
```

```
09690  67a5  ===> (opt.) Farbquelle einlesen <===
09691  67a5
09692  67a5  20 32 9e          jsr incolr       Farbquelle (0-3) einlesen
09693  67a8
09694  67a8  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
09695  67ab  d0 03             bne draw10       kein Trennzeichen
09696  67ad
09697  67ad  4c fb 9b          jmp gplot        Test/einfachen/dopp. Punkt setzen
09698  67b0
09699  67b0
09700  67b0  20 86 03   draw10 jsr chrgot       letztes Zeichen erneut nach ac
09701  67b3  c9 2c             cmp #','
09702  67b5  f0 05             beq draw20
09703  67b7
09704  67b7  c9 a4             cmp #_tot        TO Token
09705  67b9  f0 01             beq draw20
09706  67bb
09707  67bb  60                rts
09708  67bc
09709  67bc
09710  67bc  48         draw20 pha              letztes Zeichen
09711  67bd
09712  67bd  20 80 03          jsr chrget       nächstes Zeichen nach ac
09713  67c0  a2 04             ldx #xdest-vwork Zielkoord.
09714  67c2  20 70 9e          jsr incord       Start-/Zielkoord. einlesen
09715  67c5
09716  67c5  68                pla              letzt. Zeich. (TO Token = MINUS)
09717  67c6  10 06             bpl draw30       einzelnen Punkt setzen
09718  67c8
09719  67c8  ===> Strecke zeichnen <===
09720  67c8
09721  67c8  20 30 9b          jsr drawln       Strecke zeichnen
09722  67cb  4c b0 67          jmp draw10
09723  67ce
09724  67ce
09725  67ce  ===> einzelnen Punkt setzen <===
09726  67ce
09727  67ce  20 f2 9d   draw30 jsr dstpos       Zielkoord. werden akt. Koord.
09728  67d1  20 fb 9b          jsr gplot        Test/einfachen/dopp. Punkt setzen
09729  67d4  4c b0 67          jmp draw10
09730  67d7
09731  67d7
09732  67d7  ===> BASIC-Statement CHAR ($e0) <===
09733  67d7
09734  67d7  20 32 9e   char   jsr incolr       Farbquelle (0-3) einlesen
09735  67da
09736  67da  a2 29             ldx #_col40+1    Vorgabe Spalten 40 Zeichen BS
09737  67dc  a0 1a             ldy #_rows+1     Vorgabe BS Zeilen
09738  67de
09739  67de  a5 d8             lda graphm       Flag Text/Graphik/Split-BS
09740  67e0  d0 05             bne char05
09741  67e2
09742  67e2  ===> Text Modus <===
09743  67e2
09744  67e2  20 ed ff          jsr kscror       Bildschirmorganisation lesen
                                                (Sprungv.)
09745  67e5  e8                inx              Spalten +1
09746  67e6  c8                iny              Zeilen +1
```

```
09747  67e7
09748  67e7  8e 5e 11   char05 stx colcnt       max. BS Spalten +1
09749  67ea  8c 5f 11          sty rowcnt       max. BS Zeilen +1
09750  67ed
09751  67ed
09751  67ed  ===> BS Spalte CHAR einlesen <===
09752  67ed
09753  67ed  20 09 88          jsr combyt       Kommatest, 8 Bit INT nach xr
09754  67f0
09755  67f0  ec 5e 11          cpx colcnt       max. BS Spalten +1
09756  67f3  b0 0b             bcs charer       Behandlung »?illegal quantity error«
09757  67f5
09758  67f5  8e 5e 11          stx colcnt       akt. BS Spalte CHAR
09759  67f8
09760  67f8  ===> BS Zeile CHAR einlesen <===
09761  67f8
09762  67f8  20 09 88          jsr combyt       Kommatest, 8 Bit INT nach xr
09763  67fb
09764  67fb  ec 5f 11          cpx rowcnt       max. BS Zeilen +1
09765  67fe  90 03             bcc char10
09766  6800
09767  6800  ===> Behandlung »?illegal quantity error« <===
09768  6800
09769  6800  4c 28 7d   charer jmp fcerr        Behandlung »?illegal quantity error«
09770  6803
09771  6803
09772  6803  8e 5f 11   char10 stx rowcnt       akt. BS Zeile CHAR
09773  6806
09774  6806  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
09775  6809  d0 04             bne char15       kein Trennzeichen
09776  680b
09777  680b  a9 00             lda #0           String Länge
09778  680d  f0 06             beq char20       immer
09779  680f
09780  680f
09781  680f  20 5c 79   char15 jsr chkcom       Test auf folg. »Komma« sonst Fehler
09782  6812  20 7b 87          jsr sav13        Ausdruck in Arbeitsstring, frestr
09783  6815
09784  6815  8d 03 ff   char20 sta lcrc         ROM / PGM-RAM einschalten
09785  6818  8d 6e 11          sta numcnt       String Länge
09786  681b
09787  681b  98                tya              ADDRH String Descriptor
09788  681c  48                pha
09789  681d
09790  681d  8a                txa              ADDRL String Descriptor
09791  681e  48                pha
09792  681f
09793  681f  ===> REVERS Parameter einlesen <===
09794  681f
09795  681f  20 1c 9e          jsr optzer       opt. 8 Bit INT nach xr, sonst 0
09796  6822
09797  6822  8a                txa              Bit0
09798  6823  6a                ror a            nach Bit7
09799  6824  6e 3d 11          ror xysgn        RVS Flag
09800  6827
09801  6827   7.......  1=REVERS, 0=NORMAL
09802  6827
09803  6827  68                pla
```

```
09804  6828  85 24            sta index1       ADDRL String Descriptor
09805  682a
09806  682a  68               pla
09807  682b  85 25            sta index1+1     ADDRH String Descriptor
09808  682d
09809  682d  a5 d8            lda graphm       Flag Text/Graphik/Split-BS
09810  682f  d0 32            bne char40       Graphik Modus
09811  6831
09812  6831  ===> CHAR im Text Modus <===
09813  6831
09814  6831  ae 5f 11         ldx rowcnt       akt. BS Zeile CHAR
09815  6834  ac 5e 11         ldy colcnt       akt. BS Spalte CHAR
09816  6837  18               clc              Flag Schreiben
09817  6838  20 8d 92         jsr cplot        Cursor-Pos. (rel. BS-Fenster)
                                                les./schreib.
09818  683b
09819  683b  a0 00            ldy #0           Zeiger in String
09820  683d
09821  683d  2c 3d 11         bit xysgn        RVS Flag
09822  6840  10 05            bpl char25
09823  6842
09824  6842  ===> RVS ON <===
09825  6842
09826  6842  a9 12            lda #_rvson
09827  6844  20 0c c0         jsr jprint       BS Ausgabe ac
09828  6847
09829  6847  cc 6e 11  char25 cpy numcnt       String Länge
09830  684a  f0 0c            beq char30       Test / RVS OFF
09831  684c
09832  684c  20 b7 03         jsr ind1_1       LDA (index1),y aus VAR-RAM
09833  684f  20 45 a8         jsr mapio        I/O-Chips einblenden
09834  6852  20 0c c0         jsr jprint       BS Ausgabe ac
09835  6855
09836  6855  c8               iny              Zeiger in String
09837  6856  d0 ef            bne char25       immer
09838  6858
09839  6858
09840  6858  ===> Test / RVS OFF <===
09841  6858
09842  6858  2c 3d 11  char30 bit xysgn        RVS Flag
09843  685b  10 05            bpl char35       nicht gesetzt
09844  685d
09845  685d  ===> RVS OFF <===
09846  685d
09847  685d  a9 92            lda #_rvsof
09848  685f  20 0c c0         jsr jprint       BS Ausgabe ac
09849  6862
09850  6862  60         char35 rts
09851  6863
09852  6863
09853  6863  ===> CHAR im Graphik Modus <===
09854  6863
09855  6863  20 74 a0  char40 jsr isgrap       Test GRAPHIC-RAM reserviert
09856  6866
09857  6866  ad ec 11         lda upgrap       ADDRH Char.gen. Groß / Graphik
09858  6869  8d 68 11         sta charom
09859  686c
09860  686c  a5 86            lda forgnd       Vorgabe Vordergrundfarbe
```

```
09861  686e  aa              tax             Vorgabe Vordergrundfarbe
09862  686f  48              pha
09863  6870
09864  6870  a5 83           lda colsel      Farbquelle (0-3)
09865  6872  48              pha
09866  6873
09867  6873  24 d8           bit graphm      Flag Text/Graphik/Split-BS
09868  6875  10 0e           bpl char45      nicht MULTICOLOR
09869  6877
09870  6877  ===> Multicolor <===
09871  6877
09872  6877  68              pla
09873  6878  f0 16           beq char50      Farbquelle Hintergrund
09874  687a
09875  687a  4a              lsr a
09876  687b  f0 13           beq char50      Farbquelle =1
09877  687d
09878  687d  ===> Farbquelle Zusatzfarbe <===
09879  687d
09880  687d  a6 84           ldx mulco1      Zusatzfarbe 1
09881  687f  90 0f           bcc char50
09882  6881
09883  6881  a6 85           ldx mulco2      Zusatzfarbe 2
09884  6883  b0 0b           bcs char50      immer
09885  6885
09886  6885
09887  6885  ===> nicht MULTICOLOR <===
09888  6885
09889  6885  a6 86     char45 ldx forgnd      Vorgabe Vordergrundfarbe
09890  6887  68               pla
09891  6888  d0 06           bne char50
09892  688a
09893  688a  ===> Farbquelle Hintergrund <===
09894  688a
09895  688a  20 45 a8          jsr mapio       I/O-Chips einblenden
09896  688d  ae 21 d0          ldx vicbkg      VIC 8564 Hintergrundfarbe
09897  6890
09898  6890  86 86     char50 stx forgnd      Farbcode Vordergrund
09899  6892
09900  6892  ae 5f 11          ldx rowcnt      BS Zeile CHAR
09901  6895
09902  6895  a0 00             ldy #0
09903  6897  8c 60 11          sty strcnt      Zeiger in String
09904  689a
09905  689a  ac 60 11  char55 ldy strcnt       Zeiger in String
09906  689d  ee 60 11          inc strcnt      Zeiger in String
09907  68a0
09908  68a0  20 b7 03          jsr ind1_1      LDA (index1),y aus VAR-RAM
09909  68a3  8d 03 ff          sta lcrc        ROM / PGM-RAM einschalten
09910  68a6
09911  68a6  ce 6e 11          dec numcnt      Zeichenzähler
09912  68a9  30 2c             bmi char80      fertig
09913  68ab
09914  68ab  c9 0e             cmp #_uplo      Code für Groß/Kleinschrift
09915  68ad  d0 05             bne char60
09916  68af
09917  68af  ===> Umschalten Char.gen. Groß/Klein <===
09918  68af
```

```
09919  68af  ad eb 11          lda upplow       ADDRH Char.gen. Klein / Groß
09920  68b2  d0 07             bne char65       immer
09921  68b4
09922  68b4
09923  68b4  c9 8e      char60 cmp #_upgrp      Code für Groß/Graphik
09924  68b6  d0 08             bne char70
09925  68b8
09926  68b8
09926  68b8  ===> Umschalten Char.gen. Groß/Klein <===
09927  68b8
09928  68b8  ad ec 11          lda upgrap       ADDRH Grafik-Zeichengenerator
09929  68bb
09930  68bb  8d 68 11   char65 sta charom
09931  68be  d0 09             bne char75       immer
09932  68c0
09933  68c0
09934  68c0  ac 5e 11   char70 ldy colcnt       BS Spalte CHAR
09935  68c3  20 db 68          jsr chrdsp       Zeichen in Bitspeicher bringen
09936  68c6
09937  68c6  ee 5e 11          inc colcnt       BS Spalte CHAR
09938  68c9
09939  68c9  c0 27      char75 cpy #_col40-1    Spalten 40 Zeichen BS -1
09940  68cb  90 cd             bcc char55
09941  68cd
09942  68cd  a0 00             ldy #0
09943  68cf  8c 5e 11          sty colcnt       BS Spalte CHAR
09944  68d2
09945  68d2  e8                inx              BS Zeile
09946  68d3  e0 18             cpx #_rows-1     BS Zeilen -1
09947  68d5  90 c3             bcc char55
09948  68d7
09949  68d7  68         char80 pla              Zeichenfarbe
09950  68d8  85 86             sta forgnd       Farbcode Vordergrund
09951  68da  60                rts
09952  68db
09953  68db
09954  68db  ===> Zeichen in Bitspeicher bringen <===
09955  68db
09956  68db  48         chrdsp pha              Zeichen
09957  68dc  20 70 9c          jsr docolr       Farbquelle in Color RAM eintragen
09958  68df
09959  68df   AC: Ausgabezeichen
09960  68df   XR: BS Zeile
09961  68df   YR: BS Spalte
09962  68df
09963  68df  98                tya              BS Spalte CHAR
09964  68e0  18                clc
09965  68e1  7d 33 c0          adc ldtb2,x      Tab. ADDRL BS Zeilen
09966  68e4  85 8c             sta grapnt       ADDRL Byte
09967  68e6
09968  68e6  bd 4c c0          lda ldtb1,x      Tab. ADDRH BS Zeilen
09969  68e9
09970  68e9  69 00             adc #0           ADDRH Byte
09971  68eb
09972  68eb  ===> Adresse in GRAPHIC RAM berechnen <===
09973  68eb
09974  68eb  06 8c             asl grapnt       ADDRL Zeiger in Graphik-RAM
09975  68ed  2a                rol a
```

```
09976  68ee
09977  68ee  06 8c           asl grapnt       ADDRL Zeiger in Graphik-RAM
09978  68f0  2a              rol a
09979  68f1
09980  68f1  06 8c           asl grapnt       ADDRL in GRAPHIC RAM
09981  68f3  2a              rol a
09982  68f4
09983  68f4  85 8d           sta grapnt+1     ADDRH in GRAPHIC RAM
09984  68f6  8d 03 ff        sta lcrc         ROM / PGM-RAM einschalten
09985  68f9
09986  68f9  ===> Adresse Zeichen in Char.gen. berechnen <===
09987  68f9
09988  68f9  a9 00           lda #0           vorbesetzen
09989  68fb  85 77           sta mltus1       rel. ADDRH Zeichen in Char.gen.
09990  68fd
09991  68fd  68              pla              Zeichen
09992  68fe  48              pha              Zeichen
09993  68ff
09994  68ff  0a              asl a            Bit7 (SHIFT BIT)
09995  6900  26 77           rol mltus1       nach rel. ADDRH Zeichen in Char.gen.
09996  6902
09997  6902  0a              asl a            Commodore ASC nach BS Code
09998  6903
09999  6903  0a              asl a            rel. ADDRL Zeichen in Char.gen.
10000  6904  26 77           rol mltus1       rel. ADDRH Zeichen in Cahr.gen.
10001  6906
10002  6906  85 26           sta index2       abs. ADDRL Zeichen in Char.gen.
10003  6908
10004  6908  a5 77           lda mltus1       rel. ADDRH Zeichen in Char.gen.
10005  690a  6d 68 11        adc charom
10006  690d  85 27           sta index2+1     abs. ADDRH Zeichen in Char.gen.
10007  690f
10008  690f  98              tya              akt. BS Spalte CHAR
10009  6910  48              pha              akt. BS Spalte CHAR
10010  6911
10011  6911  a0 07           ldy #7           Zähler acht Char.gen. Bytes
10012  6913  ad 3d 11  movchr lda xysgn        RVS Flag
10013  6916
10014  6916   7.......  1=RVS ON
10015  6916
10016  6916  0a              asl a            Bit7 nach CARRY
10017  6917
10018  6917  b1 26           lda (index2),y   Char.gen. Byte
10019  6919  90 02           bcc movch0
10020  691b
10021  691b  ===> RVS ON: Char.gen. Byte invertieren <===
10022  691b
10023  691b  49 ff           eor #%11111111
10024  691d
10025  691d  24 d8     movch0 bit graphm       Flag Text/Graphik/Split-BS
10026  691f  10 2b           bpl movch4       nicht MULTICOLOR
10027  6921
10028  6921  ===> MULTICOLOR HI RES <===
10029  6921
10030  6921  29 aa           and #%10101010
10031  6923  85 77           sta mltus1       Char.gen. Byte
10032  6925
10033  6925  a5 83           lda colsel       Farbquelle (0-3)
```

```
10034  6927  d0 0f            bne movch1
10035  6929
10036  6929  ===> Farbquelle Hintergrund <===
10037  6929
10038  6929  a5 77            lda mltus1       Char.gen. Byte
10039  692b  b0 07            bcs movrvs       RVS ON
10040  692d
10041  692d
10041  692d  ===> RVS OFF MULTICOLOR HI RES <===
10042  692d
10043  692d  4a               lsr a
10044  692e  45 77            eor mltus1
10045  6930  49 aa            eor #%10101010
10046  6932  d0 18            bne movch4       immer
10047  6934
10048  6934
10049  6934  ===> RVS ON MULTICOLOR HI RES <===
10050  6934
10051  6934  09 55      movrvs ora #%01010101
10052  6936  d0 14            bne movch4       immer
10053  6938
10054  6938
10055  6938  c9 02      movch1 cmp #2           Farbquelle
10056  693a  d0 04            bne movch2
10057  693c
10058  693c  ===> Farbquelle Zusatzfarbe 1 <===
10059  693c
10060  693c  a5 77            lda mltus1       Char.gen. Byte
10061  693e  b0 0c            bcs movch4       immer
10062  6940
10063  6940
10064  6940  90 07      movch2 bcc movch3       Farbquelle =1
10065  6942
10066  6942  ===> Farbquelle Zusatzfarbe 2 <===
10067  6942
10068  6942  a5 77            lda mltus1       Char.gen. Byte
10069  6944  4a               lsr a
10070  6945  45 77            eor mltus1
10071  6947  90 03            bcc movch4       immer
10072  6949
10073  6949
10074  6949  ===> Farbquelle Vordergrund <===
10075  6949
10076  6949  a5 77      movch3 lda mltus1       Char.gen. Byte
10077  694b  4a               lsr a
10078  694c
10079  694c
10080  694c  ===> Char.gen. Byte nach GRAPHIC RAM <===
10081  694c
10082  694c  91 8c      movch4 sta (grapnt),y   Char.gen. Byte nach GRAPHIC RAM
10083  694e
10084  694e  88               dey              Zähler für Bytes / Zeichen
10085  694f  10 c2            bpl movchr
10086  6951
10087  6951  68               pla
10088  6952  a8               tay              akt. BS Spalte CHAR
10089  6953
10090  6953  68               pla              akt. Ausgabezeichen CHAR
```

```
10091  6954  60               rts
10092  6955
10093  6955
10094  6955  ===> BASIC-Statement LOCATE ($e6) <===
10095  6955
10096  6955  20 74 a0   locate jsr isgrap      Test GRAPHIC-RAM reserviert
10097  6958
10098  6958
10098  6958  ===> Koord. einlesen <===
10099  6958
10100  6958  a2 04            ldx #4
10101  695a  20 70 9e         jsr incord      Start-/Zielkoord. einlesen
10102  695d  4c f2 9d         jmp dstpos      Zielkoord. werden akt. Koord.
10103  6960
10104  6960
10105  6960  ===> BASIC-Statement SCALE ($e9) <===
10106  6960
10107  6960  20 f4 87   scale  jsr getbyt      8 Bit INT üb. frmnum nach xr
10108  6963  e0 02            cpx #2
10109  6965  90 03            bcc scal10
10110  6967
10111  6967  ===> Behandlung »?illegal quantity error« <===
10112  6967
10113  6967  4c 28 7d         jmp fcerr       Behandlung »?illegal quantity error«
10114  696a
10115  696a
10116  696a  8e 6a 11   scal10 stx scalem      SCALE Flag
10117  696d
10118  696d  20 86 03         jsr chrgot      letztes Zeichen erneut nach ac
10119  6970  d0 14            bne scal30      kein Trennzeichen
10120  6972
10121  6972  ===> Vorgaben SCALE einstellen (1023/1023) <===
10122  6972
10123  6972  a2 00            ldx #0
10124  6974  a9 50            lda #>65535/1023*320
10125  6976  a0 32            ldy #>65535/1023*200
10126  6978
10127  6978  24 d8            bit graphm      Flag Text/Graphik/Split-BS
10128  697a  10 01            bpl scal20      nein
10129  697c
10130  697c  4a               lsr a           511
10131  697d
10132  697d
10133  697d  ===> Vorgaben SCALE abspeichern <===
10134  697d
10135  697d  86 87      scal20 stx scalex      SCALE Low-Byte x-Wert (Abszisse)
10136  697f  85 88            sta scalex+1    SCALE High-Byte x-Wert (Abszisse)
10137  6981
10138  6981  86 89            stx scaley      SCALE Low-Byte y-Wert (Ordinate)
10139  6983  84 8a            sty scaley+1    SCALE High-Byte y-Wert (Ordinate)
10140  6985
10141  6985  60               rts
10142  6986
10143  6986
10144  6986  ===> SCALE x-Wert einlesen/berechnen <===
10145  6986
10146  6986  20 c4 69   scal30 jsr gtlt8k      Kommatest, pos. Zahl <8K nach FAC
10147  6989
```

```
10148  6989  a9 d8           lda #<n320       GK-Konst. 320*65535
10149  698b  a0 69           ldy #>n320       GK-Konst. 320*65535
10150  698d  20 89 8a        jsr romupk       ROM Konst. ac/yr nach ARG
10151  6990  20 4c 8b        jsr fdivt        Division ARG durch FAC
10152  6993
10153  6993  20 15 88        jsr getadr       pos. FAC als 16 Bit INT linnum,
                                              yr/ac
10154  6996  c9 00           cmp #0           High-Byte
10155  6998  d0 04           bne scal40
10156  699a
10157  699a  c0 00           cpy #0           Low-Byte
10158  699c  f0 37           beq badval
10159  699e
10160  699e  48        scal40 pha             High-Byte SCALE Abszisse
10161  699f
10162  699f  98               tya
10163  69a0  48               pha             Low-Byte SCALE Abszisse
10164  69a1
10165  69a1  ===> SCALE y-Wert einlesen/berechnen <===
10166  69a1
10167  69a1  20 c4 69        jsr gtlt8k       Kommatest, pos. Zahl <8K nach FAC
10168  69a4
10169  69a4  a9 dd           lda #<n200       GK-Konst. 200*65535
10170  69a6  a0 69           ldy #>n200       GK-Konst. 200*65535
10171  69a8  20 89 8a        jsr romupk       ROM Konst. ac/yr nach ARG
10172  69ab  20 4c 8b        jsr fdivt        Division ARG durch FAC
10173  69ae
10174  69ae  20 15 88        jsr getadr       pos. FAC als 16 Bit INT linnum,
                                              yr/ac
10175  69b1  c9 00           cmp #0           High-Byte
10176  69b3  d0 04           bne scal50       Eingaben SCALE abspeichern
10177  69b5
10178  69b5  c0 00           cpy #0           Low-Byte
10179  69b7  f0 1c           beq badval
10180  69b9
10181  69b9
10182  69b9  ===> Eingaben SCALE abspeichern <===
10183  69b9
10184  69b9  84 89      scal50 sty scaley      SCALE Low-Byte y-Wert (Ordinate)
10185  69bb  85 8a             sta scaley+1    SCALE High-Byte y-Wert (Ordinate)
10186  69bd
10187  69bd  68               pla
10188  69be  85 87            sta scalex      SCALE Low-Byte x-Wert (Abszisse)
10189  69c0
10190  69c0  68               pla
10191  69c1  85 88            sta scalex+1    SCALE High-Byte x-Wert (Abszisse)
10192  69c3
10193  69c3  60               rts
10194  69c4
10195  69c4
10196  69c4  ===> Kommatest, pos. Zahl <8K nach FAC <===
10197  69c4
10198  69c4  20 5c 79   gtlt8k jsr chkcom      Test auf folg. »Komma« sonst Fehler
10199  69c7  20 d7 77         jsr frmnum       folg. num. Variable nach FAC
10200  69ca
10201  69ca  a5 68            lda facsgn       FAC: SG
10202  69cc  30 07            bmi badval
10203  69ce
```

```
10204  69ce  a5 63            lda facexp       FAC: EX
10205  69d0  c9 90            cmp #_expcn+15   2 hoch 15
10206  69d2  b0 01            bcs badval
10207  69d4
10208  69d4  60               rts
10209  69d5
10210  69d5
10211  69d5
10211  69d5  ===> Behandlung »?illegal quantity error« <===
10212  69d5
10213  69d5  4c 28 7d   badval jmp fcerr       Behandlung »?illegal quantity error«
10214  69d8
10215  69d8
10216  69d8  ===> GK-Konst. 320*65535 <===
10217  69d8
10218  69d8  99 1f ff   n320   .byte $99, $1f, $ff, $60, $00
10218  69db  60 00
10219  69dd
10220  69dd
10221  69dd  ===> GK-Konst. 200*65535 <===
10222  69dd
10223  69dd  98 47 ff   n200   .byte $98, $47, $ff, $38, $00
10223  69e0  38 00
10224  69e2
10225  69e2
10226  69e2                   .end
10227  69e2                   .lib basic.graph_3
10228  69e2  put"@basic.graph_3"
```

```
10230  69e2
10231  69e2
10232  69e2  ===> BASIC-Statement COLOR ($e7) <===
10233  69e2
10234  69e2  20 f4 87   color  jsr getbyt       Farbquelle (0-6) nach xr
10235  69e5
10236  69e5   0 = Hintergrundfarbe (40 Zeichen BS)
10237  69e5   1 = Vordergrundfarbe (graph.)
10238  69e5   2 = Zusatzfarbe 1
10239  69e5   3 = Zusatzfarbe 2
10240  69e5   4 = Rahmenfarbe
10241  69e5   5 = Textfarbe
10242  69e5   6 = Hintergrundfarbe (80 Zeichen BS)
10243  69e5
10244  69e5  e0 07            cpx #7
10245  69e7  b0 60            bcs sclerr
10246  69e9
10247  69e9  86 77            stx mltus1       Farbquelle
10248  69eb
10249  69eb  ===> Farbnummer (1-16) einlesen <===
10250  69eb
10251  69eb  20 09 88         jsr combyt       Kommatest, 8 Bit INT nach xr
10252  69ee
10253  69ee   1 = schwarz              9 = hellbraun
10254  69ee   2 = weiß               10 = braun
10255  69ee   3 = rot                 11 = rosa
10256  69ee   4 = grün                12 = dunkelgrau
10257  69ee   5 = violett             13 = grau
10258  69ee   6 = dunkelgrün          14 = hellgrün
10259  69ee   7 = blau                15 = hellblau
10260  69ee   8 = gelb                16 = hellgrau
10261  69ee
10262  69ee  ca               dex
10263  69ef  e0 10            cpx #16
10264  69f1  b0 56            bcs sclerr
10265  69f3
10266  69f3  20 45 a8         jsr mapio        I/O-Chips einblenden
10267  69f6  8a               txa              Farbnummer
10268  69f7
10269  69f7  a6 77            ldx mltus1       Farbquelle
10270  69f9  e0 01            cpx #1
10271  69fb  f0 07            beq colr10       Vordergrundfarbe eintragen
10272  69fd  b0 09            bcs colr20
10273  69ff
10274  69ff  ===> Hintergrundfarbe (40 Zeichen BS) eintragen <===
10275  69ff
10276  69ff  8d 21 d0         sta vicbkg       VIC 8564 Hintergrundfarbe
10277  6a02  d0 3f            bne colr80       immer
10278  6a04
10279  6a04
10280  6a04  ===> Vordergrundfarbe eintragen <===
10281  6a04
10282  6a04  85 86      colr10 sta forgnd      Vordergrundfarbe
10283  6a06  f0 3b             beq colr80      immer
10284  6a08
10285  6a08
10286  6a08  e0 03      colr20 cpx #3
10287  6a0a  f0 06             beq colr30      Zusatzfarbe 2 eintragen
```

```
10288  6a0c  b0 08            bcs colr40
10289  6a0e
10290  6a0e  ===> Zusatzfarbe 1 eintragen <===
10291  6a0e
10292  6a0e  85 84            sta mulco1      Multi-Color Farbe 1
10293  6a10  d0 31            bne colr80      immer
10294  6a12
10295  6a12
10296  6a12  ===> Zusatzfarbe 2 eintragen <===
10297  6a12
10298  6a12  85 85      colr30 sta mulco2      Zusatzfarbe 2
10299  6a14  f0 2d            beq colr80      immer
10300  6a16
10301  6a16
10302  6a16  e0 05      colr40 cpx #5
10303  6a18  f0 07            beq colr50      Textfarbe eintragen
10304  6a1a  b0 16            bcs colr70      Hintergrundfarbe (80 Zeichen BS)
                                              eintr.
10305  6a1c
10306  6a1c  ===> Rahmenfarbe eintragen <===
10307  6a1c
10308  6a1c  8d 20 d0         sta vicbrd      VIC 8564 Rahmenfarbe
10309  6a1f  d0 22            bne colr80      immer
10310  6a21
10311  6a21
10312  6a21  ===> Textfarbe eintragen <===
10313  6a21
10314  6a21  24 d7      colr50 bit mode       BS-Modus 40/80-Zeichen
10315  6a23  10 08            bpl colr60      40 Zeichen BS
10316  6a25
10317  6a25  ===> 80 Zeichen BS <===
10318  6a25
10319  6a25  aa               tax
10320  6a26
10321  6a26  a5 f1            lda attrib      akt. Attribut
10322  6a28  29 f0            and #%11110000  Farbcode löschen
10323  6a2a  1d 4c 6a         ora adjust,x    Tab. Farbcodes VDC 8563
10324  6a2d
10325  6a2d
10326  6a2d  ===> 40 Zeichen BS <===
10327  6a2d
10328  6a2d  85 f1      colr60 sta attrib      akt. Attribut
10329  6a2f  4c 43 6a         jmp colr80
10330  6a32
10331  6a32
10332  6a32  ===> Hintergrundfarbe (80 Zeichen BS) eintr. <===
10333  6a32
10334  6a32  aa         colr70 tax            Farbnummer
10335  6a33
10336  6a33  a9 1a            lda #26         Hintergrundfarbe VDC 8563
10337  6a35
10338  6a35   7654....  Vordergrundfarbe, wenn Bit6 Reg. 25 = 0
10339  6a35   ....3210  Hintergrundfarbe alle Zeichen und Rahmenfarbe
10340  6a35
10341  6a35  8d 00 d6         sta vdcadr      VDC 8563 Adreß-Register
10342  6a38
10343  6a38  ad 01 d6         lda vdcdat      VDC 8563 Daten-Register
10344  6a3b  29 f0            and #%11110000  Farbcode Hintergrund löschen
```

```
10345  6a3d  1d 4c 6a          ora adjust,x     Tab. Farbcodes VDC 8563
10346  6a40  8d 01 d6          sta vdcdat       VDC 8563 Daten-Register
10347  6a43
10348  6a43  20 5c 6a   colr80 jsr setpcl       Vordergr.fb. zu Hintergr./Zusatzf. 1
10349  6a46  4c 1e 9e          jmp optbyt       opt. 8 Bit INT nach xr
10350  6a49
10351  6a49
10352  6a49  ===> Behandlung »?illegal quantity error« <===
10353  6a49
10354  6a49  4c 28 7d   sclerr jmp fcerr        Behandlung »?illegal quantity error«
10355  6a4c
10356  6a4c
10357  6a4c  ===> Tab. Farbcodes VDC 8563 <===
10358  6a4c
10359  6a4c  00         adjust .byte %00000000  schwarz
10360  6a4d  0f                .byte %00001111  weiß
10361  6a4e  08                .byte %00001000  rot
10362  6a4f  07                .byte %00000111  grün
10363  6a50  0b                .byte %00001011  violett
10364  6a51  04                .byte %00000100  dunkelgrün
10365  6a52  02                .byte %00000010  blau
10366  6a53  0d                .byte %00001101  gelb
10367  6a54  0a                .byte %00001010  hellbraun
10368  6a55  0c                .byte %00001100  braun
10369  6a56  09                .byte %00001001  rosa
10370  6a57  06                .byte %00000110  dunkelgrau
10371  6a58  01                .byte %00000001  grau
10372  6a59  05                .byte %00000101  hellgrün
10373  6a5a  03                .byte %00000011  hellblau
10374  6a5b  0e                .byte %00001110  hellgrau
10375  6a5c
10376  6a5c
10377  6a5c  ===> Vordergr.fb. zu Hintergr./Zusatzf. 1 <===
10378  6a5c
10379  6a5c  a5 86      setpcl lda forgnd       Farbcode Vordergrund
10380  6a5e  0a                asl a            nach Bit7-4
10381  6a5f  0a                asl a
10382  6a60  0a                asl a
10383  6a61  0a                asl a
10384  6a62  85 77             sta mltus1       Farbcode Vordergrund
10385  6a64
10386  6a64  20 45 a8          jsr mapio        I/O-Chips einblenden
10387  6a67
10388  6a67  ===> Vordergrundfarbe zu Hintergrundfarbe VIC <===
10389  6a67
10390  6a67  ad 21 d0          lda vicbkg       VIC 8564 Hintergrundfarbe
10391  6a6a  29 0f             and #%00001111   Farbcode Vordergr. löschen
10392  6a6c  05 77             ora mltus1       Farbcode Vordergrund
10393  6a6e  8d e2 03          sta fgbg         VIC Vorder-/Hintergrund Farbe
10394  6a71
10395  6a71  ===> Vordergrundfarbe zu Zusatzfarbe 1 <===
10396  6a71
10397  6a71  a5 84             lda mulco1       Multi-Color Farbe 1
10398  6a73  05 77             ora mltus1       Farbcode Vordergrund
10399  6a75  8d e3 03          sta fgmc1        Zusatzfarbe 1
10400  6a78
10401  6a78  60                rts
10402  6a79
```

```
10403  6a79
10404  6a79
10404  6a79  ===> BASIC-Statement SCNCLR ($e8) <===
10405  6a79
10406  6a79  d0 14       scnclr bne scncl1       SCNCLR Modus einlesen
10407  6a7b
10408  6a7b  20 8c 81           jsr unpgpm       Graphik Modus 40/80 Zeichen BS
                                                 berechnen
10409  6a7e
10410  6a7e    0/5  Text                                  GRAPH 0
10411  6a7e    1/6  HI-RES                                GRAPH 1
10412  6a7e    2/7  geteilter BS HI-RES/Text              GRAPH 2
10413  6a7e    3/8  HI-RES Multi-Color                   GRAPH 3
10414  6a7e    4/9  geteilter BS HI-RES Multi-Color/Text  GRAPH 4
10415  6a7e
10416  6a7e  c9 05              cmp #5
10417  6a80  90 09              bcc sc40_0
10418  6a82
10419  6a82  ===> 80 Zeichen BS <===
10420  6a82
10421  6a82  e9 05              sbc #5
10422  6a84  f0 58              beq clr80        80 Zeichen BS TEXT löschen
10423  6a86
10424  6a86  ===> 80 Zeichen BS TEXT / HI RES löschen <===
10425  6a86
10426  6a86  48                 pha              Graphik Modus
10427  6a87  20 de 6a           jsr clr80        80 Zeichen BS TEXT löschen
10428  6a8a  68                 pla              Graphik Modus
10429  6a8b
10430  6a8b  aa         sc40_0  tax              Graphik Modus
10431  6a8c  4c 9b 6a           jmp sclr40       40 Zeichen BS löschen
10432  6a8f
10433  6a8f
10434  6a8f  ===> SCNCLR Modus einlesen <===
10435  6a8f
10436  6a8f  20 f4 87   scncl1  jsr getbyt       8 Bit INT üb. frmnum nach xr
10437  6a92
10438  6a92   0 = 40 Zeichen BS
10439  6a92   1 = HI RES GRAPHIC
10440  6a92   2 = HI RES SPLIT
10441  6a92   3 = MULTICOLOR
10442  6a92   4 = MULTICOLOR SPLIT
10443  6a92   5 = 80 Zeichen BS
10444  6a92
10445  6a92  e0 05      scncl2  cpx #5           Graphik/SCNCLR Modus
10446  6a94  f0 48              beq clr80        80 Zeichen BS TEXT löschen
10447  6a96  90 03              bcc sclr40       40 Zeichen BS löschen
10448  6a98
10449  6a98  ===> Behandlung »?illegal quantity error« <===
10450  6a98
10451  6a98  4c 28 7d           jmp fcerr        Behandlung »?illegal quantity error«
10452  6a9b
10453  6a9b
10454  6a9b  ===> 40 Zeichen BS löschen <===
10455  6a9b
10456  6a9b  8a         sclr40  txa              Graphik/SCNCLR Modus
10457  6a9c  f0 54              beq clr40        40 Zeichen BS TEXT löschen
10458  6a9e
```

```
10459  6a9e
10459  6a9e  ===> 40 Zeichen BS GRAPHIC löschen <===
10460  6a9e
10461  6a9e  20 74 a0          jsr isgrap       Test GRAPHIC-RAM reserviert
10462  6aa1
10463  6aa1  8a                txa              Graphik/SCNCLR Modus
10464  6aa2  48                pha
10465  6aa3
10466  6aa3  29 01             and #1           ungerade testen
10467  6aa5  d0 21             bne sc40_2       nicht TEXT
10468  6aa7
10469  6aa7  20 f2 6a          jsr clr40        40 Zeichen BS TEXT löschen
10470  6aaa
10471  6aaa  a5 d7             lda mode         BS-Modus 40/80-Zeichen
10472  6aac
10473  6aac   00000000  40 Zeichen BS
10474  6aac   10000000  80 Zeichen BS
10475  6aac
10476  6aac  48                pha              BS Modus 40/80 Zeichen
10477  6aad  10 03             bpl sc40_1       40 Zeichen BS
10478  6aaf
10479  6aaf  20 5f ff          jsr kswapr       Umschalten auf 40 Zeichen BS
10480  6ab2
10481  6ab2
10482  6ab2  ===> 1. BS Zeile Textfenster berechnen <===
10483  6ab2
10484  6ab2  ad 34 0a   sc40_1 lda split        VIC Rasterwert geteilter BS
10485  6ab5  38                sec
10486  6ab6  e9 30             sbc #48
10487  6ab8  4a                lsr a
10488  6ab9  4a                lsr a
10489  6aba  4a                lsr a
10490  6abb  aa                tax              1 BS Zeile Textfenster
10491  6abc
10492  6abc  a0 00             ldy #0           BS Spalte Textfenster
10493  6abe  18                clc
10494  6abf  20 8d 92          jsr cplot        Curs.Pos. rel. BS Fenster schreiben
10495  6ac2
10496  6ac2  68                pla              BS Modus 40/80 Zeichen
10497  6ac3  10 03             bpl sc40_2       40 Zeichen BS
10498  6ac5
10499  6ac5  20 5f ff          jsr kswapr       Umschalten auf 80 Zeichen BS
10500  6ac8
10501  6ac8  68         sc40_2 pla              Grafik/SCNCLR Modus
10502  6ac9  29 02             and #%00000010
10503  6acb  f0 03             beq sc40_3       nicht MULTICOLOR
10504  6acd
10505  6acd  20 17 6b          jsr clrhcl       MULTICOLOR RAM löschen
10506  6ad0
10507  6ad0  20 30 6b   sc40_3 jsr clrgrf       GRAPHIC RAM und Farb RAM löschen
10508  6ad3
10509  6ad3  ===> PIXEL CURS. HOME Position <===
10510  6ad3
10511  6ad3  a9 00             lda #0
10512  6ad5  a2 03             ldx #3
10513  6ad7  9d 31 11   sc40_4 sta xypos,x      Koord. PIXEL CURS.
10514  6ada  ca                dex
10515  6adb  10 fa             bpl sc40_4
```

```
10516  6add
10517  6add  60               rts
10518  6ade
10519  6ade
10520  6ade  ===> 80 Zeichen BS TEXT löschen <===
10521  6ade
10522  6ade  a5 d7      clr80  lda mode         BS-Modus 40/80-Zeichen
10523  6ae0  48               pha
10524  6ae1  30 03            bmi cl80_1       80 Zeichen BS
10525  6ae3
10526  6ae3  20 5f ff         jsr kswapr       Umschalten auf 80 Zeichen BS
10527  6ae6
10528  6ae6  a9 93      cl80_1 lda #_clr
10529  6ae8  20 69 92         jsr cbsout       ind. Ausgabe ac auf akt. Kanal
10530  6aeb
10531  6aeb  68               pla              BS Modus 40/80 Zeichen
10532  6aec  30 03            bmi cl80_2       80 Zeichen BS
10533  6aee
10534  6aee  20 5f ff         jsr kswapr       Umschalten auf 40 Zeichen BS
10535  6af1
10536  6af1  60         cl80_2 rts
10537  6af2
10538  6af2
10539  6af2  ===> 40 Zeichen BS TEXT löschen <===
10540  6af2
10541  6af2  a5 d7      clr40  lda mode         BS-Modus 40/80-Zeichen
10542  6af4  48               pha
10543  6af5  10 03            bpl cl40_1       40 Zeichen BS
10544  6af7
10545  6af7  20 5f ff         jsr kswapr       Umschalten auf 40 Zeichen BS
10546  6afa
10547  6afa  a9 93      cl40_1 lda #_clr
10548  6afc  20 69 92         jsr cbsout       ind. Ausgabe ac auf akt. Kanal
10549  6aff
10550  6aff  68               pla              BS Modus 40/80 Zeichen
10551  6b00  10 03            bpl cl40_2       40 Zeichen BS
10552  6b02
10553  6b02  20 5f ff         jsr kswapr       Umschalten auf 80 Zeichen BS
10554  6b05  60         cl40_2 rts
10555  6b06
10556  6b06
10557  6b06  ===> xr Pages mit Kode ac ab Page yr füllen <===
10558  6b06
10559  6b06  84 8d      scnfil sty grapnt+1    ADDRH Zeiger in Graphik-RAM
10560  6b08
10561  6b08  a0 00            ldy #0
10562  6b0a  84 8c            sty grapnt       ADDRL Zeiger in Graphik-RAM
10563  6b0c
10564  6b0c  91 8c      scnflp sta (grapnt),y  ADDRL Zeiger in Graphik-RAM
10565  6b0e  88               dey
10566  6b0f  d0 fb            bne scnflp
10567  6b11
10568  6b11  e6 8d            inc grapnt+1     ADDRH Zeiger in Graphik-RAM
10569  6b13  ca               dex              Seiten Zähler
10570  6b14  d0 f6            bne scnflp
10571  6b16
10572  6b16  60               rts
10573  6b17
```

```
10574  6b17
10575  6b17
10575  6b17  ===> MULTICOLOR RAM löschen <===
10576  6b17
10577  6b17  20 45 a8   clrhcl jsr mapio        I/O-Chips einblenden
10578  6b1a  78                 sei
10579  6b1b
10580  6b1b  a5 01              lda r8502        CPU 8502 Port Data Reg.
10581  6b1d  48                 pha
10582  6b1e
10583  6b1e  29 fe              and #%11111110
10584  6b20
10585  6b20   xxxxxxx0  MULTICOLOR RAM ein $d800-$dbff
10586  6b20   xxxxxxx1  BASIC ROM C 64 ein $a000-$bfff
10587  6b20
10588  6b20  85 01              sta r8502        CPU 8502 Port Data Reg.
10589  6b22
10590  6b22  a5 85              lda mulco2       Zusatzfarbe 2
10591  6b24  a0 d8              ldy #>colrmh     ADDRH MULTICOLOR RAM
10592  6b26  a2 04              ldx #4           4 Seiten
10593  6b28  20 06 6b           jsr scnfil       xr Pages mit Kode ac ab Page yr
                                                 füllen
10594  6b2b
10595  6b2b  68                 pla              CPU 8502 Port Data Reg.
10596  6b2c  85 01              sta r8502        CPU 8502 Port Data Reg.
10597  6b2e
10598  6b2e  58                 cli
10599  6b2f  60                 rts
10600  6b30
10601  6b30
10602  6b30  ===> GRAPHIC RAM und Farb RAM löschen <===
10603  6b30
10604  6b30  a9 00      clrgrf  lda #0           Löschcode GRAPHIC RAM
10605  6b32  a0 20              ldy #>gpbase     ADDRH GRAPHIC RAM
10606  6b34  a2 20              ldx #32          32 Seiten
10607  6b36  20 06 6b           jsr scnfil       xr Pages mit Kode ac ab Page yr
                                                 füllen
10608  6b39
10609  6b39  ad e2 03           lda fgbg         Löschcode FARB RAM
10610  6b3c
10611  6b3c  24 d8              bit graphm       Flag Text/Graphik/Split-BS
10612  6b3e  10 03              bpl clrgf1       nicht MULTICOLOR
10613  6b40
10614  6b40  ad e3 03           lda fgmc1        Löschcode MULTICOLOR
10615  6b43
10616  6b43  20 45 a8   clrgf1  jsr mapio        I/O-Chips einblenden
10617  6b46
10618  6b46  a0 1c              ldy #>colrml     ADDRH COLOR RAM lo
10619  6b48  a2 04              ldx #4           4 Seiten
10620  6b4a  20 06 6b           jsr scnfil       xr Pages mit Kode ac ab Page yr
                                                 füllen
10621  6b4d
10622  6b4d  a2 3f              ldx #$3f
10623  6b4f  a0 07              ldy #7
10624  6b51  8a         clrgf2  txa              Spritezeiger
10625  6b52  99 f8 1f           sta gpbase-8,y   letzte 8 Bytes
10626  6b55  ca                 dex
10627  6b56  88                 dey
```

```
10628  6b57  10 f8            bpl clrgf2
10629  6b59
10630  6b59  60               rts
10631  6b5a
10632  6b5a
10633  6b5a  ===> BASIC-Statement GRAPHIC ($de) <===
10634  6b5a
10635  6b5a  c9 9c      grphik cmp #_clrt       CLR Token
10636  6b5c  d0 0b             bne grph10       GRAPHIC Modus einschalten
10637  6b5e
10638  6b5e  ===> GRAPHIC CLR <===
10639  6b5e
10640  6b5e  20 22 a0          jsr clrhir       Test / GRAPHIC RAM freigeben
10641  6b61  20 80 03          jsr chrget       nächstes Zeichen nach ac
10642  6b64
10643  6b64  a9 00             lda #0
10644  6b66  85 d8             sta graphm       Flag Text/Graphik/Split-BS
10645  6b68  60                rts
10646  6b69
10647  6b69
10648  6b69  ===> GRAPHIC Modus einschalten <===
10649  6b69
10650  6b69  20 f4 87   grph10 jsr getbyt       8 Bit INT üb. frmnum nach xr
10651  6b6c
10652  6b6c   0 = 40 Zeichen BS TEXT
10653  6b6c   1 = HI RES
10654  6b6c   2 = HI RES SPLIT BS
10655  6b6c   3 = MULTICOLOR
10656  6b6c   4 = MULTICOLOR SPLIT BS
10657  6b6c   5 = 80 Zeichen BS TEXT
10658  6b6c
10659  6b6c  8a                txa              GRAPHIC Modus
10660  6b6d  48                pha              GRAPHIC Modus
10661  6b6e
10662  6b6e  e0 05             cpx #5
10663  6b70  f0 42             beq grph60       80 Zeichen BS TEXT Modus
10664  6b72  b0 4d             bcs graper       Behandlung »?illegal quantity error«
10665  6b74
10666  6b74  bd c4 6b          lda modes,x      Tab. Code Nibbles GRAPHIC Modi
10667  6b77  85 d8             sta graphm       Flag Text/Graphik/Split-BS
10668  6b79  f0 07             beq grph20       40 Zeichen BS Text
10669  6b7b
10670  6b7b  20 4f 9f          jsr sethir       Test / GRAPHIC RAM einrichten
10671  6b7e
10672  6b7e  24 d8             bit graphm
10673  6b80  50 07             bvc grph40       kein Textfenster
10674  6b82
10675  6b82
10676  6b82  ===> 40 Zeichen BS Text <===
10677  6b82
10678  6b82  24 d7      grph20 bit mode         BS-Modus 40/80-Zeichen
10679  6b84  10 03             bpl grph40       40 Zeichen BS
10680  6b86
10681  6b86  20 5f ff   grph30 jsr kswapr       Umschalten 40/80 Zeichen Bildschirm
                                                (flip) (Sprungv.)
10682  6b89
10683  6b89  20 1c 9e   grph40 jsr optzer       opt. 8 Bit INT nach xr, sonst 0
10684  6b8c
```

```
10685  6b8c   0 = BS nicht löschen
10686  6b8c   1 = BS löschen
10687  6b8c
10688  6b8c  e0 02            cpx #2
10689  6b8e  b0 31            bcs graper      Behandlung »?illegal quantity error«
10690  6b90
10691  6b90  8a               txa             Lösch Modus
10692  6b91  48               pha             Lösch Modus
10693  6b92
10694  6b92  ===> 1. Zeile TEXT Fenster einlesen <===
10695  6b92
10696  6b92  a2 14            ldx #20         Vorgabe 1. Zeile TEXT Fenster
10697  6b94  20 1e 9e         jsr optbyt      opt. 8 Bit INT nach xr
10698  6b97
10699  6b97  e0 1a            cpx #26
10700  6b99  b0 26            bcs graper      Behandlung »?illegal quantity error«
10701  6b9b
10702  6b9b  ===> Rasterwert 1. Zeile TEXT Fenster berechnen <===
10703  6b9b
10704  6b9b  8a               txa             1. Zeile TEXT Fenster
10705  6b9c  0a               asl a
10706  6b9d  0a               asl a
10707  6b9e  0a               asl a
10708  6b9f  69 30            adc #48
10709  6ba1  8d 34 0a         sta split       VIC Rasterwert geteilter BS
10710  6ba4
10711  6ba4  68               pla             Lösch Modus
10712  6ba5  a8               tay
10713  6ba6
10714  6ba6  68               pla             GRAPHIC Modus
10715  6ba7  aa               tax
10716  6ba8
10717  6ba8  98               tya             Lösch Modus
10718  6ba9  f0 03            beq grph50      SCALE aus
10719  6bab
10720  6bab  20 92 6a         jsr scncl2
10721  6bae
10722  6bae
10723  6bae  ===> SCALE aus <===
10724  6bae
10725  6bae  a9 00      grph50 lda #0
10726  6bb0  8d 6a 11         sta scalem      SCALE Flag
10727  6bb3  60               rts
10728  6bb4
10729  6bb4
10730  6bb4  ===> 80 Zeichen BS TEXT Modus <===
10731  6bb4
10732  6bb4  24 d7      grph60 bit mode        BS-Modus 40/80-Zeichen
10733  6bb6  30 d1            bmi grph40      80 Zeichen BS
10734  6bb8
10735  6bb8  ===> 40 Zeichen BS <===
10736  6bb8
10737  6bb8  a5 d8            lda graphm      Flag Text/Graphik/Split-BS
10738  6bba  29 bf            and #%10111111  SPLIT BS Bit löschen
10739  6bbc  85 d8            sta graphm      Flag Text/Graphik/Split-BS
10740  6bbe
10741  6bbe  4c 86 6b         jmp grph30
10742  6bc1
```

```
10743  6bc1
10744  6bc1
10744  6bc1  ===> Behandlung »?illegal quantity error« <===
10745  6bc1
10746  6bc1  4c 28 7d   graper jmp fcerr        Behandlung »?illegal quantity error«
10747  6bc4
10748  6bc4
10749  6bc4  ===> Tab. Code Nibbles GRAPHIC Modi <===
10750  6bc4
10751  6bc4  00         modes  .byte %00000000  TEXT
10752  6bc5  20                .byte %00100000  HI RES
10753  6bc6  60                .byte %01100000  HI RES SPLIT
10754  6bc7  a0                .byte %10100000  MULTICOLOR
10755  6bc8  e0                .byte %11100000  MULTICOLOR SPLIT
10756  6bc9
10757  6bc9
10758  6bc9  ===> DISK-B.-Statement BANK ($fe $02) <===
10759  6bc9
10760  6bc9  20 f4 87   bank   jsr getbyt       8 Bit INT üb. frmnum nach xr
10761  6bcc  e0 10             cpx #16
10762  6bce  b0 04             bcs bnkerr       Behandlung »?illegal quantity error«
10763  6bd0
10764  6bd0  8e d5 03          stx curbnk       akt. Bank
10765  6bd3  60                rts
10766  6bd4
10767  6bd4
10768  6bd4  ===> Behandlung »?illegal quantity error« <===
10769  6bd4
10770  6bd4  4c 28 7d   bnkerr jmp fcerr        Behandlung »?illegal quantity error«
10771  6bd7
10772  6bd7
10773  6bd7  ===> BASIC-Statement SLEEP ($fe $0b) <===
10774  6bd7
10775  6bd7  20 12 88   sleep  jsr getwrd       16 Bit INT nach linnum
10776  6bda  a2 00             ldx #0
10777  6bdc  78                sei
10778  6bdd
10779  6bdd  8c 1d 0a          sty slpcnt       Low-Byte Sekunden
10780  6be0  8d 1e 0a          sta slpcnt+1     High-Byte Sekunden
10781  6be3  8e 1f 0a          stx slpcnt+2     Timer für SLEEP
10782  6be6
10783  6be6  ===> Sekunden mal 60 (60 IRQ/s) <===
10784  6be6
10785  6be6  20 0c 6c          jsr sltim2       Timer mal zwei
10786  6be9  20 16 6c          jsr xayslp       Timer + xr, ac, yr
10787  6bec  20 09 6c          jsr sltim4       Timer mal vier
10788  6bef
10789  6bef  ac 1d 0a          ldy slpcnt       Timer für SLEEP
10790  6bf2  ad 1e 0a          lda slpcnt+1     Timer für SLEEP
10791  6bf5  ae 1f 0a          ldx slpcnt+2     Timer für SLEEP
10792  6bf8  20 09 6c          jsr sltim4       Timer mal vier
10793  6bfb
10794  6bfb  20 16 6c          jsr xayslp       Timer + xr, ac, yr
10795  6bfe  58                cli
10796  6bff
10797  6bff
```

```
10798  6bff  ===> Warten auf TIMER Unterlauf <===
10799  6bff
10800  6bff  20 b5 4b   sleplp jsr istpky       Stopp-Taste abfragen
10801  6c02  ae 1f 0a          ldx slpcnt+2     Timer für SLEEP
10802  6c05  e8                inx
10803  6c06  d0 f7             bne sleplp       Warten auf TIMER Unterlauf
10804  6c08
10805  6c08  60                rts
10806  6c09
10807  6c09
10808  6c09  ===> Timer mal vier <===
10809  6c09
10810  6c09  20 0c 6c   sltim4 jsr sltim2       Timer mal zwei
10811  6c0c
10812  6c0c
10813  6c0c  ===> Timer mal zwei <===
10814  6c0c
10815  6c0c  0e 1d 0a   sltim2 asl slpcnt       Timer für SLEEP
10816  6c0f  2e 1e 0a          rol slpcnt+1     Timer für SLEEP
10817  6c12  2e 1f 0a          rol slpcnt+2     Timer für SLEEP
10818  6c15  60                rts
10819  6c16
10820  6c16
10821  6c16  ===> Timer + xr, ac, yr <===
10822  6c16
10823  6c16  48         xayslp pha
10824  6c17
10825  6c17  98                tya
10826  6c18  6d 1d 0a          adc slpcnt       Timer für SLEEP
10827  6c1b  8d 1d 0a          sta slpcnt       Timer für SLEEP
10828  6c1e
10829  6c1e  68                pla
10830  6c1f  6d 1e 0a          adc slpcnt+1     Timer für SLEEP
10831  6c22  8d 1e 0a          sta slpcnt+1     Timer für SLEEP
10832  6c25
10833  6c25  8a                txa
10834  6c26  6d 1f 0a          adc slpcnt+2     Timer für SLEEP
10835  6c29  8d 1f 0a          sta slpcnt+2     Timer für SLEEP
10836  6c2c
10837  6c2c  60                rts
10838  6c2d
10839  6c2d
10840  6c2d  ===> BASIC-Statement WAIT ($92) <===
10841  6c2d
10842  6c2d  20 03 88   wait   jsr getnum       16 Bit INT nach linnum, Kommatest, 8
                                                Bit INT nach xr
10843  6c30  86 4b             stx lstpnt       AND Parameter
10844  6c32
10845  6c32  a2 00             ldx #0           Vorgabe XOR Parameter
10846  6c34  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
10847  6c37  f0 03             beq wait10       Trennzeichen
10848  6c39
10849  6c39  20 09 88          jsr combyt       Kommatest, 8 Bit INT nach xr
10850  6c3c  86 4c      wait10 stx lstpnt+1     XOR Parameter
10851  6c3e
10852  6c3e
```

```
10853  6c3e  ===> Bitmuster testen, warten <===
10854  6c3e
10855  6c3e  a0 00      waitlp ldy #0          Zeiger
10856  6c40  ae d5 03          ldx curbnk      akt. Bank
10857  6c43  a9 16             lda #<linnum    Speicherzelle
10858  6c45  20 74 ff          jsr kifetv      LDA (FETVEC),y von Bank in xr
                                               (Sprungv.)
10859  6c48
10860  6c48  45 4c             eor lstpnt+1
10861  6c4a  25 4b             and lstpnt
10862  6c4c  f0 f0             beq waitlp      Bitmuster testen, warten
10863  6c4e
10864  6c4e  60                rts
10865  6c4f
10866  6c4f
10867  6c4f                    .end
10868  6c4f                    .lib basic.sprite_1
10869  6c4f  put"@basic.sprite_1"
```

```
10871  6c4f
10872  6c4f
10873  6c4f  ===> BASIC-Statement SPRITE ($fe $07) <===
10874  6c4f
10875  6c4f  20 bb 6c   sprite jsr gtsprn       Sprite Nummer -1 nach mltus1
10876  6c52
10877  6c52  ===> (opt.) Modus ein/aus einlesen <===
10878  6c52
10879  6c52  20 1e 9e          jsr optbyt       opt. 8 Bit INT nach xr
10880  6c55  90 05             bcc sprit1       keine Eingabe
10881  6c57
10882  6c57  ===> Sprite ein-/ausschalten <===
10883  6c57
10884  6c57  a0 15             ldy #21          VIC 8564 Sprite Enable Register
10885  6c59
10886  6c59   76543210  Sprite7-0 (1=0ein)
10887  6c59
10888  6c59  20 9b 6c          jsr sprbit       Sprite-Bit in VIC-Register YR
                                                setzen/löschen
10889  6c5c
10890  6c5c
10891  6c5c  ===> (opt.) Sprite Farbe einlesen <===
10892  6c5c
10893  6c5c  20 1e 9e   sprit1 jsr optbyt       opt. 8 Bit INT nach xr
10894  6c5f  90 0e             bcc sprit2       keine Eingabe
10895  6c61
10896  6c61  ===> Sprite Farbe setzen <===
10897  6c61
10898  6c61  ca                dex
10899  6c62  e0 10             cpx #16
10900  6c64  b0 32             bcs sprerr       Behandlung »?illegal quantity error«
10901  6c66
10902  6c66  8a                txa              Sprite Farbe
10903  6c67  a6 77             ldx mltus1       Sprite Nummer -1
10904  6c69  20 45 a8          jsr mapio        I/O-Chips einblenden
10905  6c6c  9d 27 d0          sta vicspc,x     VIC 8564 Sprite Farbe
10906  6c6f
10907  6c6f
10908  6c6f  ===> (opt.) Priorität einlesen <===
10909  6c6f
10910  6c6f  20 1e 9e   sprit2 jsr optbyt       opt. 8 Bit INT nach xr
10911  6c72  90 05             bcc sprit3       keine Eingabe
10912  6c74
10913  6c74  a0 1b             ldy #27          Prioritäts Register
10914  6c76
10915  6c76   76543210  Priorität Sprite7-0 (1=ein)
10916  6c76
10917  6c76  20 9b 6c          jsr sprbit       Sprite-Bit in VIC-Register YR
                                                setzen/löschen
10918  6c79
10919  6c79
10920  6c79  ===> (opt.) Dehnung in x-Richt. einl. <===
10921  6c79
10922  6c79  20 1e 9e   sprit3 jsr optbyt       opt. 8 Bit INT nach xr
10923  6c7c  90 05             bcc sprit4       keine Eingabe
10924  6c7e
10925  6c7e  a0 1d             ldy #29          Expand Horizontal Register
10926  6c80
```

```
10927  6c80   76543210  Sprite7-0 horiz. verdopp. (1=ein)
10928  6c80
10929  6c80  20 9b 6c          jsr sprbit       Sprite-Bit in VIC-Register YR
                                                setzen/löschen
10930  6c83
10931  6c83
10932  6c83  ===> (opt.) Dehnung in y-Richt. einl. <===
10933  6c83
10934  6c83  20 1e 9e   sprit4 jsr optbyt       opt. 8 Bit INT nach xr
10935  6c86  90 05              bcc sprit5       keine Eingabe
10936  6c88
10937  6c88  a0 17              ldy #23          Expand Vertical Register
10938  6c8a
10939  6c8a   76543210  Sprite7-0 vert. verdopp. (1=ein)
10940  6c8a
10941  6c8a  20 9b 6c          jsr sprbit       Sprite-Bit in VIC-Register YR
                                                setzen/löschen
10942  6c8d
10943  6c8d
10944  6c8d  ===> (opt.) Modus einlesen <===
10945  6c8d
10946  6c8d  20 1e 9e   sprit5 jsr optbyt       opt. 8 Bit INT nach xr
10947  6c90  90 05              bcc sprit6       keine Eingabe
10948  6c92
10949  6c92  a0 1c              ldy #28          MULTICOLOR Register
10950  6c94
10951  6c94   76543210  Sprite7-0 MULTICOLOR (1=ein)
10952  6c94
10953  6c94  20 9b 6c          jsr sprbit       Sprite-Bit in VIC-Register YR
                                                setzen/löschen
10954  6c97
10955  6c97  60         sprit6 rts
10956  6c98
10957  6c98
10958  6c98  ===> Behandlung »?illegal quantity error« <===
10959  6c98
10960  6c98  4c 28 7d   sprerr jmp fcerr        Behandlung »?illegal quantity error«
10961  6c9b
10962  6c9b
10963  6c9b  ===> Sprite-Bit in VIC-Register YR setzen/löschen <===
10964  6c9b
10965  6c9b  8a         sprbit txa              Eingabe Byte
10966  6c9c  4a                lsr a            Bit Wert nach CARRY
10967  6c9d  d0 f9              bne sprerr       größer 1
10968  6c9f
10969  6c9f  a6 77              ldx mltus1       Sprite Nummer -1
10970  6ca1  bd b3 6c           lda sbits,x      Tabelle Maskenbits Bit0-7
10971  6ca4  20 45 a8           jsr mapio        I/O-Chips einblenden
10972  6ca7
10973  6ca7  ===> Bit setzen <===
10974  6ca7
10975  6ca7  19 00 d0           ora vic,y        VIC 8564 Basisadresse
10976  6caa  b0 03              bcs sprbt1       Eingabe = 1
10977  6cac
10978  6cac  ===> Bit löschen <===
10979  6cac
10980  6cac  5d b3 6c           eor sbits,x      Tabelle Maskenbits Bit0-7
10981  6caf
```

```
10982  6caf  99 00 d0   sprbt1 sta vic,y        und in VIC-Register schreiben
10983  6cb2  60               rts
10984  6cb3
10985  6cb3
10986  6cb3  ===> Tabelle Maskenbits Bit0-7 <===
10987  6cb3
10988  6cb3  01 02 04   sbits  .byte %00000001, %00000010, %00000100, %00001000
10988  6cb6  08
10989  6cb7
10990  6cb7  10 20 40          .byte %00010000, %00100000, %01000000, %10000000
10990  6cba  80
10991  6cbb
10992  6cbb
10993  6cbb  ===> Sprite Nummer -1 nach mltus1 <===
10994  6cbb
10995  6cbb  20 f4 87   gtsprn jsr getbyt       8 Bit INT üb. frmnum nach xr
10996  6cbe  ca               dex
10997  6cbf  e0 08            cpx #8
10998  6cc1  b0 d5            bcs sprerr       Behandlung »?illegal quantity error«
10999  6cc3
11000  6cc3  86 77            stx mltus1
11001  6cc5  60               rts
11002  6cc6
11003  6cc6
11004  6cc6  ===> BASIC-Statement MOVSPR ($fe $06) <===
11005  6cc6
11006  6cc6  20 bb 6c   movspr jsr gtsprn       Sprite Nummer -1 nach mltus1
11007  6cc9
11008  6cc9  ===> Abszisse od. Winkel Ziel einlesen <===
11009  6cc9
11010  6cc9  20 9e 6d          jsr sprcor       Abszisse od. Winkel Ziel nach yr/ac
11011  6ccc
11012  6ccc  2c 6e 11          bit numcnt       Test Komma nach Sprite Nummer
11013  6ccf
11014  6ccf   .6......  0 = Komma gelesen
11015  6ccf
11016  6ccf  50 03            bvc movs05       Winkel bzw. Abszisse Ziel eintr.
11017  6cd1
11018  6cd1  ===> Behandlung »?syntax error« <===
11019  6cd1
11020  6cd1  4c 6c 79          jmp snerr        Behandlung »?syntax error«
11021  6cd4
11022  6cd4
11023  6cd4  ===> Winkel bzw. Abszisse Ziel eintr. <===
11024  6cd4
11025  6cd4  8c 35 11   movs05 sty xdest        Low-Byte
11026  6cd7  8c 37 11          sty ydest        Low-Byte
11027  6cda
11028  6cda  8d 36 11          sta xdest+1      High-Byte
11029  6cdd  8d 38 11          sta ydest+1      High-Byte
11030  6ce0
11031  6ce0  ===> Ordinate Ziel od. Geschwind. (0-15) einlesen <===
11032  6ce0
11033  6ce0  20 9e 6d          jsr sprcor       Ordinate Ziel od. Geschwind. nach
                                                yr/ac
11034  6ce3
11035  6ce3  2c 6e 11          bit numcnt       Zeichenzähler
11036  6ce6
```

```
11037  6ce6   7.......  1 = Koord., 0 = Geschwind. gelesen
11038  6ce6   .6......  0 = Komma gelesen
11039  6ce6
11040  6ce6  50 61             bvc movs35       Ordinate Ziel eintragen
11041  6ce8  30 3a             bmi movs25       rel. (skal. od. Pol.) Koord. Ziel
                                                 eintr.
11042  6cea
11043  6cea  ===> Geschwindigkeit eintragen <===
11044  6cea
11045  6cea  98                tya              Low-Byte = Geschwindigkeit (0-15)
11046  6ceb  48                pha              Geschwindigkeit
11047  6cec
11048  6cec  a0 04             ldy #4
11049  6cee  20 74 9a          jsr getand       SIN/COS Winkel (yr) nach sinpc/cospc
11050  6cf1
11051  6cf1  a6 77             ldx mltus1       Sprite Nummer -1
11052  6cf3  bc d9 6d          ldy sproff,x     Tabelle Offsets für Parameter
                                                 Sprite0-7
11053  6cf6
11054  6cf6  a9 00             lda #0
11055  6cf8  99 7e 11          sta sspeed,y     Konstante Schritte / BASIC-IRQ
11056  6cfb
11057  6cfb  c8                iny              Zähler für Schritte/BASIC IRQ
11058  6cfc
11059  6cfc  ===> SIN und COS halbieren <===
11060  6cfc
11061  6cfc  a2 03             ldx #3
11062  6cfe  5e 4a 11   movs10 lsr sinval,x
11063  6d01  ca                dex
11064  6d02  7e 4a 11          ror sinval,x
11065  6d05  ca                dex
11066  6d06  10 f6             bpl movs10
11067  6d08
11068  6d08
11069  6d08  ===> berechn. Quadr./SIN/COS nach Sprite Par. <===
11070  6d08
11071  6d08  e8         movs15 inx
11072  6d09  bd 49 11          lda angsgn,x     Quadrant
11073  6d0c  c8                iny
11074  6d0d  99 7e 11          sta sspeed,y     Konstante Schritte / BASIC-IRQ
11075  6d10
11076  6d10  e0 04             cpx #4
11077  6d12  d0 f4             bne movs15       berechn. Quadr./SIN/COS nach Sprite
                                                 Par.
11078  6d14
11079  6d14  ===> ? Koord. Ziel ? <===
11080  6d14
11081  6d14  a9 00             lda #0
11082  6d16  c8         movs20 iny
11083  6d17  99 7e 11          sta sspeed,y     Konstante Schritte / BASIC-IRQ
11084  6d1a  ca                dex
11085  6d1b  d0 f9             bne movs20
11086  6d1d
11087  6d1d  ===> Geschwindigkeit eintragen <===
11088  6d1d
11089  6d1d  68                pla              neue Sprite Geschwindigkeit
11090  6d1e  29 0f             and #%00001111   0-15
11091  6d20  99 74 11          sta t3,y
```

```
11092  6d23
11093  6d23  60              rts
11094  6d24
11095  6d24
11096  6d24  ===> rel. (skal. od. Pol.) Koord. Ziel eintr. <===
11097  6d24
11098  6d24  20 39 81   movs25 jsr swapxy       Inhalt xr / yr tauschen
11099  6d27  a8               tay              Low-Byte nach yr
11100  6d28  8a               txa              High-Byte nach ac
11101  6d29  20 77 9a         jsr gtang1       SIN/COS ac/yr nach sinpc/cospc
11102  6d2c
11103  6d2c  a2 04            ldx #xdest-vwork
11104  6d2e  20 4a 9d         jsr scalxy       Test/SCALE: Koord. umrechnen
11105  6d31
11106  6d31  a2 04            ldx #xdest-vwork
11107  6d33  18               clc              Flag für SIN
11108  6d34
11109  6d34
11110  6d34  ===> Koord. Ziel berech./eintr. <===
11111  6d34
11112  6d34  20 ce 9a   movs30 jsr angmlt       Koord.(xr) mal SIN/COS nach ac/yr
11113  6d37
11114  6d37  9d 31 11         sta xdest-4,x
11115  6d3a  98               tya
11116  6d3b  9d 32 11         sta xdest+1-4,x
11117  6d3e
11118  6d3e  e8               inx
11119  6d3f  e8               inx
11120  6d40
11121  6d40  e0 06            cpx #6
11122  6d42  f0 f0            beq movs30       Flag für COS
11123  6d44
11124  6d44  6e 6e 11         ror numcnt       Zeichenzähler
11125  6d47
11126  6d47   7.......  1 = Prod. positiv, 0 = negativ
11127  6d47   .6......  1 = Koord., 0 = Geschwind. gelesen
11128  6d47   ..5.....  0 = Komma gelesen
11129  6d47
11130  6d47  30 0b            bmi movs40       Produkt positiv
11131  6d49
11132  6d49
11133  6d49  ===> Ordinate Ziel eintragen <===
11134  6d49
11135  6d49  8c 37 11   movs35 sty ydest
11136  6d4c  8d 38 11         sta ydest+1
11137  6d4f
11138  6d4f  a2 04            ldx #xdest-vwork
11139  6d51  20 4a 9d         jsr scalxy       Test/SCALE: Koord. umrechnen
11140  6d54
11141  6d54  a5 77      movs40 lda mltus1       Sprite Nummer -1
11142  6d56  aa               tax              Zeiger auf Low-Byte
11143  6d57
11144  6d57  0a               asl a
11145  6d58  a8               tay              Zeiger auf High-Byte
11146  6d59
11147  6d59  ad 37 11         lda ydest
11148  6d5c
11149  6d5c  0e 6e 11         asl numcnt       Zeichenzähler
```

```
11150  6d5f
11151  6d5f   7.......  1 = Prod. positiv, 0 = negativ
11152  6d5f   .6......  1 = Koord., 0 = Geschwind. gelesen
11153  6d5f   ..5.....  0 = Komma gelesen
11154  6d5f
11155  6d5f  90 09             bcc movs50
11156  6d61
11157  6d61  18                clc
11158  6d62  10 03             bpl movs45       Geschwindigkeit
11159  6d64
11160  6d64  49 ff             eor #%11111111   Vorzeichenwechsel
11161  6d66
11162  6d66  38                sec
11163  6d67  79 d7 11   movs45 adc postab+1,y   Ordinate Sprite
11164  6d6a  78         movs50 sei
11165  6d6b  99 d7 11          sta postab+1,y   Ordinate Sprite
11166  6d6e
11167  6d6e  ad 35 11          lda xdest
11168  6d71
11169  6d71  0e 6e 11          asl numcnt       Zeichenzähler
11170  6d74  10 12             bpl movs60
11171  6d76
11172  6d76  18                clc
11173  6d77  79 d6 11          adc postab,y     Abszisse Sprite
11174  6d7a  99 d6 11          sta postab,y     Abszisse Sprite
11175  6d7d
11176  6d7d  b0 03             bcs movs55
11177  6d7f  ee 36 11          inc xdest+1
11178  6d82
11179  6d82  ad e6 11   movs55 lda spmsb        MSB-Position Sprite7-0
11180  6d85  4c 91 6d          jmp movs65
11181  6d88
11182  6d88
11183  6d88  99 d6 11   movs60 sta postab,y     Abszisse Sprite
11184  6d8b
11185  6d8b  ad e6 11          lda spmsb        MSB-Position Sprite7-0
11186  6d8e  1d b3 6c          ora sbits,x      Tabelle Maskenbits Bit0-7
11187  6d91
11188  6d91  4e 36 11   movs65 lsr xdest+1
11189  6d94  b0 03             bcs movs70
11190  6d96
11191  6d96  5d b3 6c          eor sbits,x      Sprite Bit löschen
11192  6d99
11193  6d99  8d e6 11   movs70 sta spmsb        MSB-Position Sprite7-0
11194  6d9c
11195  6d9c  58                cli
11196  6d9d  60                rts
11197  6d9e
11198  6d9e
11199  6d9e  ===> Koord. / Geschw. Sprite einlesen <===
11200  6d9e
11201  6d9e  20 c6 6d   sprcor jsr comchk       Test auf folg. Komma, CARRY=0
11202  6da1
11203  6da1  6e 6e 11          ror numcnt       CARRY nach Bit7
11204  6da4  10 0b             bpl sprco1       Komma folgt
11205  6da6
11206  6da6  c9 3b             cmp #';'         Polarkoord.
11207  6da8  f0 13             beq sprco3       Bit Flag für Koord.
```

```
11208  6daa
11209  6daa  c9 23            cmp #'#'          Geschwindigkeit
11210  6dac  f0 0e            beq sprco2        Bit Flag für Geschwindigkeit
11211  6dae
11212  6dae
11212  6dae  ===> Behandlung »?syntax error« <===
11213  6dae
11214  6dae  4c 6c 79          jmp snerr         Behandlung »?syntax error«
11215  6db1
11216  6db1
11217  6db1  ===> Test auf rel. Koord. <===
11218  6db1
11219  6db1  20 86 03   sprco1 jsr chrgot       letztes Zeichen erneut nach ac
11220  6db4
11221  6db4  c9 aa             cmp #_plust       PLUS Token
11222  6db6  f0 05             beq sprco3        Bit Flag für Koord.
11223  6db8
11224  6db8  c9 ab             cmp #_minut       MINUS Token
11225  6dba  f0 01             beq sprco3        Bit Flag für Koord.
11226  6dbc
11227  6dbc
11228  6dbc  ===> Bit Flag für Geschwindigkeit <===
11229  6dbc
11230  6dbc  18         sprco2 clc
11231  6dbd  6e 6e 11   sprco3 ror numcnt        Zeichenzähler
11232  6dc0
11233  6dc0   7.......  1 = Koord., 0 = Geschwind. gelesen
11234  6dc0   .6......  0 = Komma gelesen
11235  6dc0
11236  6dc0  20 d7 77          jsr frmnum        folg. num. Variable nach FAC
11237  6dc3  4c 19 88          jmp getsad        FAC als 16 Bit INT linnum, yr/ac
11238  6dc6
11239  6dc6
11240  6dc6  ===> Test auf folg. Komma, CARRY=0 <===
11241  6dc6
11242  6dc6  20 86 03   comchk jsr chrgot       letztes Zeichen erneut nach ac
11243  6dc9  f0 0d             beq comck2        bei Trennzeichen
11244  6dcb
11245  6dcb  c9 2c             cmp #','
11246  6dcd  18                clc
11247  6dce  f0 01             beq comck1        Komma
11248  6dd0
11249  6dd0  38                sec
11250  6dd1  08         comck1 php
11251  6dd2  48                pha
11252  6dd3  20 80 03          jsr chrget        nächstes Zeichen nach ac
11253  6dd6  68                pla               letztes Zeichen
11254  6dd7  28                plp               Komma: cc, ansonsten cs
11255  6dd8  60         comck2 rts
11256  6dd9
11257  6dd9
11258  6dd9  ===> Tabelle Offsets für Parameter Sprite0-7 <===
11259  6dd9
11260  6dd9  00 0b 16   sproff .byte 0, 11, 22, 33, 44, 55, 66, 77
11260  6ddc  21 2c 37
11260  6ddf  42 4d
11261  6de1
11262  6de1
```

```
11263  6de1                    .end
11264  6de1                    .lib basic.sound_1
11265  6de1  put"@basic.sound_1"
```

```
11267  6de1
11268  6de1
11269  6de1  ===> BASIC-Statement PLAY ($fe $04) <===
11270  6de1
11271  6de1  20 7b 87   play   jsr sav13        Ausdruck in Arbeitsstring, frestr
11272  6de4  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
11273  6de7  85 77              sta mltus1       String Länge
11274  6de9
11275  6de9  20 ce 6f           jsr clrplf       Flags für +/- Halbton löschen
11276  6dec  85 78              sta hulp         String Zeiger = 0
11277  6dee
11278  6dee
11279  6dee  ===> Test / Zeichen aus String spielen <===
11280  6dee
11281  6dee  a4 78      playlp ldy hulp         String Zeiger
11282  6df0  c4 77             cpy mltus1       String Länge
11283  6df2  f0 0d             beq playrt
11284  6df4
11285  6df4  ===> Zeichen aus String spielen <===
11286  6df4
11287  6df4  20 b7 03          jsr ind1_1       LDA (index1),y aus VAR-RAM
11288  6df7  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
11289  6dfa  20 02 6e          jsr pl1chr       ein Zeichen (ac) spielen
11290  6dfd
11291  6dfd  e6 78             inc hulp         String Zeiger
11292  6dff  d0 ed             bne playlp       Test / Zeichen aus String spielen
11293  6e01
11294  6e01
11295  6e01  60         playrt rts
11296  6e02
11297  6e02
11298  6e02
11299  6e02  ===> ein Zeichen (ac) spielen <===
11300  6e02
11301  6e02  c9 20      pl1chr cmp #_spc
11302  6e04  d0 01             bne pl1c10       Test Note (a-g)
11303  6e06
11304  6e06  60                rts              Space ignorieren
11305  6e07
11306  6e07
11307  6e07  ===> Test Note (a-g) <===
11308  6e07
11309  6e07  c9 41      pl1c10 cmp #'a'
11310  6e09  90 07             bcc pl1c15       Test auf Befehle Tondauer
11311  6e0b
11312  6e0b  c9 48             cmp #'h'
11313  6e0d  b0 03             bcs pl1c15       Test auf Befehle Tondauer
11314  6e0f
11315  6e0f  4c 1e 6f          jmp plnote       Freq.Konst. Note (a-g) berechnen
11316  6e12
11317  6e12
11318  6e12  ===> Test auf Befehle Tondauer <===
11319  6e12
11320  6e12  a2 04      pl1c15 ldx #4
11321  6e14  dd e7 6f   pl1c20 cmp notes,x      Tab. Befehle Tondauer
11322  6e17  d0 03             bne pl1c25
11323  6e19
11324  6e19  4c 07 6f          jmp stnoln       Übergabe Wert Tondauer 0-4
```

```
11325  6e1c
11326  6e1c
11327  6e1c  ca          pl1c25 dex
11328  6e1d  10 f5              bpl pl1c20
11329  6e1f
11330  6e1f  ===> Test auf Befehl für PAUSE <===
11331  6e1f
11332  6e1f  c9 52              cmp #'r'
11333  6e21  d0 03              bne pl1c30       Test punktierte Note
11334  6e23
11335  6e23  4c 78 6f           jmp plrest       Pause
11336  6e26
11337  6e26
11338  6e26  ===> Test punktierte Note <===
11339  6e26
11340  6e26  c9 2e       pl1c30 cmp #'.'
11341  6e28  d0 03              bne pl1c35       Test übrige Befehle PLAY
11342  6e2a
11343  6e2a  4c 03 6f           jmp pldot        Tondauer um 50% verlängern
11344  6e2d
11345  6e2d
11346  6e2d  ===> Test übrige Befehle PLAY <===
11347  6e2d
11348  6e2d  a2 05       pl1c35 ldx #5
11349  6e2f  dd ec 6f    pl1c40 cmp mutabl,x     Tab. Befehle PLAY
11350  6e32  d0 03              bne pl1c45
11351  6e34
11352  6e34  4c 52 6f           jmp plcmd        PLAY Befehl bearbeiten
11353  6e37
11354  6e37
11355  6e37  ca          pl1c45 dex
11356  6e38  10 f5              bpl pl1c40
11357  6e3a
11358  6e3a  ===> Test auf Halbton höher (#) <===
11359  6e3a
11360  6e3a  c9 23              cmp #'#'
11361  6e3c  d0 03              bne pl1c50       Test auf Halbton tiefer ($)
11362  6e3e
11363  6e3e  4c 69 6f           jmp plshrp       Erhöhung um Halbton
11364  6e41
11365  6e41
11366  6e41  ===> Test auf Halbton tiefer ($) <===
11367  6e41
11368  6e41  c9 24       pl1c50 cmp #'$'
11369  6e43  d0 03              bne pl1c55       Test auf Ziffern 0-9
11370  6e45
11371  6e45  4c 6c 6f           jmp plflat       Erniedrigung um Halbton
11372  6e48
11373  6e48
11374  6e48  ===> Test auf Ziffern 0-9 <===
11375  6e48
11376  6e48  38          pl1c55 sec
11377  6e49  e9 30              sbc #'0'         ASCII nach HEX
11378  6e4b  c9 0a              cmp #10
11379  6e4d  90 03              bcc pl1c60       Übergabe Ziffer an PLAY Befehl
11380  6e4f
11381  6e4f  4c fd 6e           jmp pbadvl       Behandlung »?illegal quantity error«
11382  6e52
```

```
11383  6e52
11384  6e52
11384  6e52  ===> Übergabe Ziffer an PLAY Befehl <===
11385  6e52
11386  6e52  0e 26 01   pl1c60 asl plyflg       Flag PLAY Befehle
11387  6e55
11388  6e55   7.......  V Stimme     1-3
11389  6e55   .6......  O Oktave     0-6
11390  6e55   ..5.....  T HÜLLKURVE  0-9
11391  6e55   ...4....  U Lautstärke 0-9
11392  6e55   ....3...  X Filter     0-1
11393  6e55   .....210  unbenutzt
11394  6e55
11395  6e55  b0 46            bcs setvoi       Übergabe Wert 1-3 VOICE
11396  6e57
11397  6e57  0e 26 01         asl plyflg       Flag PLAY Befehle
11398  6e5a  b0 4c            bcs setoct       Übergabe Wert 0-6 OCTAVE
11399  6e5c
11400  6e5c  0e 26 01         asl plyflg       Flag PLAY Befehle
11401  6e5f  b0 50            bcs setenv       Übergabe Wert 0-9  HÜLLKURVE
11402  6e61
11403  6e61  0e 26 01         asl plyflg       Flag PLAY Befehle
11404  6e64  90 77            bcc setvol       Übergabe Wert 0-9  VOLUME
11405  6e66
11406  6e66  ===> Test 0-1 für Filter <===
11407  6e66
11408  6e66  c9 02            cmp #2
11409  6e68  90 03            bcc setfil       Übergabe Wert 0-1 FILTER
11410  6e6a
11411  6e6a  ===> PLAYFLAG rücks., Beh. »?illegal quantity error« <===0
11412  6e6a
11413  6e6a  4c fd 6e         jmp pbadvl       Behandlung »?illegal quantity error«
11414  6e6d
11415  6e6d
11416  6e6d  ===> Übergabe Wert 0-1 FILTER <===
11417  6e6d
11418  6e6d  4a         setfil lsr a            HEX Ziffer
11419  6e6e
11420  6e6e  ac 2f 12         ldy voice        VOICE Nummer -1
11421  6e71  be e4 6f         ldx times2,y     Tab. Zeigerwerte für Voice 1-3
11422  6e74
11423  6e74
11424  6e74  ===> Warten auf Ton Ende <===
11425  6e74
11426  6e74  bd 24 12   setf10 lda voichi,x     Tab. High-Bytes Tondauer (1-3)
11427  6e77  10 fb             bpl setf10       Warten auf Ton Ende
11428  6e79
11429  6e79  b9 b3 6c          lda sbits,y      Tabelle Maskenbits Bit0-7
11430  6e7c  0d 73 12          ora flters+2     Kopie SID Reg. 23 (RESNCE)
11431  6e7f
11432  6e7f   7654....  Filter Resonanz
11433  6e7f   ....3...  Filter EXTERN,  1=ein
11434  6e7f   .....2..  Filter VOICE 3, 1=ein
11435  6e7f   ......1.  Filter VOICE 2, 1=ein
11436  6e7f   .......0  Filter VOICE 1, 1=ein
11437  6e7f
11438  6e7f  b0 03            bcs setf20
11439  6e81
```

```
11440  6e81   CARRY = 1 für HEX Ziffer = 1
11441  6e81   CARRY = 0 für HEX Ziffer = 0
11442  6e81
11443  6e81  59 b3 6c          eor sbits,y      Tabelle Maskenbits Bit0-7
11444  6e84
11445  6e84  8d 73 12   setf20 sta flters+2     Filter-Modus
11446  6e87
11447  6e87  ad 74 12          lda flters+3     Kopie SID Reg. 24 (VOL/FILT)
11448  6e8a
11449  6e8a   7.......  1 = VOICE 3 aus
11450  6e8a   .6......  1 = Filter Hochpaß ein
11451  6e8a   ..5.....  1 = Filter Bandpaß ein
11452  6e8a   ...4....  1 = Filter Tiefpaß ein
11453  6e8a   ....3210  0-15 Lautstärke
11454  6e8a
11455  6e8a  8d 75 12          sta oldvol       letzter Wert SID VOL/FILT
11456  6e8d
11457  6e8d  a2 03             ldx #3
11458  6e8f  bd 71 12   setf30 lda flters,x     Kopie SID Regs. 21-24
11459  6e92  20 45 a8          jsr mapio        I/O-Chips einblenden
11460  6e95  9d 15 d4          sta sid+fclow,x  SID Filter Regs. 21-24
11461  6e98
11462  6e98  ca                dex
11463  6e99  10 f4             bpl setf30
11464  6e9b  30 5a             bmi clrflg       Flag PLAY Bef. löschen, RTS
11465  6e9d
11466  6e9d
11467  6e9d  ===> Übergabe Wert 1-3 VOICE <===
11468  6e9d
11469  6e9d  aa         setvoi tax              HEX Ziffer
11470  6e9e  ca                dex
11471  6e9f  e0 03             cpx #3
11472  6ea1  b0 5a             bcs pbadvl       Behandlung »?illegal quantity error«
11473  6ea3
11474  6ea3  ===> VOICE Nummer -1 abspeichern <===
11475  6ea3
11476  6ea3  8e 2f 12          stx voice        VOICE Nummer -1
11477  6ea6  90 4f             bcc clrflg       Flag PLAY Bef. löschen, RTS
11478  6ea8
11479  6ea8
11480  6ea8  ===> Übergabe Wert 0-6 OCTAVE <===
11481  6ea8
11482  6ea8  c9 07      setoct cmp #7
11483  6eaa  b0 51             bcs pbadvl       Behandlung »?illegal quantity error«
11484  6eac
11485  6eac  ===> OCTAVE Nummer abspeichern <===
11486  6eac
11487  6eac  8d 2b 12          sta octave       OCTAVE Nummer 0-6
11488  6eaf  90 46             bcc clrflg       Flag PLAY Bef. löschen, RTS
11489  6eb1
11490  6eb1
11491  6eb1  ===> Übergabe Wert 0-9  HÜLLKURVE <===
11492  6eb1
11493  6eb1  aa         setenv tax              HEX Ziffer
11494  6eb2
11495  6eb2
```

```
11496  6eb2  ===> HÜLLKURVE Nummer ac einstellen <===
11497  6eb2
11498  6eb2  20 45 a8   stenv1 jsr mapio        I/O-Chips einblenden
11499  6eb5  ac 2f 12          ldy voice        VOICE Nummer -1
11500  6eb8
11501  6eb8  bd 53 12          lda wavtab,x     Tab. WELLENFORMEN akt. HÜLLKURVEN
11502  6ebb  99 30 12          sta wave,y       Tab. HÜllk.werte VOICE 1-3
11503  6ebe
11504  6ebe  b9 39 70          lda offtab,y     Tab. SID Offsets VOICE 1-3
11505  6ec1  a8                tay              SID Offset VOICE
11506  6ec2  bd 3f 12          lda atktab,x     Tab. ATTACK/DECAY akt. HÜLLKURVEN
11507  6ec5  99 05 d4          sta sid+atkdcy,y SID Attack/Decay Zeiten
11508  6ec8
11509  6ec8  bd 49 12          lda sustab,x     Tab. SUSTAIN/RELEASE akt. HÜLLK.
11510  6ecb  99 06 d4          sta sid+susrel,y SID Sustain/Release Zeiten
11511  6ece
11512  6ece  bd 5d 12          lda pulslo,x     Tab. Low-Bytes PULSBREITE akt. HÜLLK.
11513  6ed1  99 02 d4          sta sid+pulsef,y SID Low-Byte PULSBREITE
11514  6ed4
11515  6ed4  bd 67 12          lda pulshi,x     Tab. High-Byte PULSBREITE akt. HÜLLK.
11516  6ed7  99 03 d4          sta sid+pulsec,y SID High-Byte PULSBREITE
11517  6eda
11518  6eda  4c f7 6e          jmp clrflg       Flag PLAY Bef. löschen, RTS
11519  6edd
11520  6edd
11521  6edd  ===> Übergabe Wert 0-9  VOLUME <===
11522  6edd
11523  6edd  aa          setvol tax             HEX Ziffer
11524  6ede
11525  6ede  ad 74 12          lda flters+3     Kopie SID Reg. 24 (VOL/FILT)
11526  6ee1  29 f0             and #%11110000
11527  6ee3
11528  6ee3   7.......  1 = VOICE 3 aus
11529  6ee3   .6......  1 = Filter Hochpaß ein
11530  6ee3   ..5.....  1 = Filter Bandpaß ein
11531  6ee3   ...4....  1 = Filter Tiefpaß ein
11532  6ee3   ....3210  0-15 Lautstärke
11533  6ee3
11534  6ee3  1d 3c 70          ora voltab,x     Tab. Werte VOLUME 0-9
11535  6ee6  8d 74 12          sta flters+3     Kopie SID Reg. 24 (VOL/FILT)
11536  6ee9
11537  6ee9  ad 75 12          lda oldvol       letzter Wert SID VOL/FILT
11538  6eec  29 f0             and #%11110000
11539  6eee  1d 3c 70          ora voltab,x     Tab. Werte VOLUME 0-9
11540  6ef1  20 45 a8          jsr mapio        I/O-Chips einblenden
11541  6ef4  8d 18 d4          sta sid+fltvol   SID Filterart / Lautstärke
11542  6ef7
11543  6ef7
11544  6ef7  ===> Flag PLAY Bef. löschen, RTS <===
11545  6ef7
11546  6ef7  a9 00       clrflg lda #0
11547  6ef9  8d 26 01          sta plyflg       Flag PLAY Befehle
11548  6efc  60                rts
11549  6efd
11550  6efd
11551  6efd  ===> Behandlung »?illegal quantity error« <===
11552  6efd
11553  6efd  20 f7 6e    pbadvl jsr clrflg      Flag PLAY Bef. löschen, RTS
```

```
11554  6f00  4c 28 7d          jmp fcerr        Behandlung »?illegal quantity error«
11555  6f03
11556  6f03
11557  6f03
11557  6f03  ===> Tondauer um 50% verlängern <===
11558  6f03
11559  6f03  8d 33 12   pldot  sta dnote        Flag für punktierte Note
11560  6f06  60                rts
11561  6f07
11562  6f07
11563  6f07  ===> Übergabe Wert Tondauer 0-4 <===
11564  6f07
11565  6f07  a0 80      stnoln ldy #<_beats     Tondauer Konst. 1/1 Note
11566  6f09  8c 29 12          sty ntime        Low-Byte Tondauer Konst.
11567  6f0c
11568  6f0c  a0 04             ldy #>_beats     Tondauer Konst. 1/1 Note
11569  6f0e  8c 2a 12          sty ntime+1      High-Byte Tondauer Konst.
11570  6f11
11571  6f11  ca         stnol1 dex              HEX Ziffer 0-4
11572  6f12  30 09             bmi stnol2
11573  6f14
11574  6f14  ===> Konstante für Tondauer halbieren <===
11575  6f14
11576  6f14  4e 2a 12          lsr ntime+1      High-Byte Tondauer Konst.
11577  6f17  6e 29 12          ror ntime        Low-Byte Tondauer Konst.
11578  6f1a
11579  6f1a  4c 11 6f          jmp stnol1
11580  6f1d
11581  6f1d
11582  6f1d  60         stnol2 rts
11583  6f1e
11584  6f1e
11585  6f1e  ===> Freq.Konst. Note (a-g) berechnen <===
11586  6f1e
11587  6f1e  38         plnote sec
11588  6f1f  e9 41             sbc #'a'         ASCII a-g nach 0-7
11589  6f21
11590  6f21  aa                tax              HEX Ziffer 0-7
11591  6f22  bd f2 6f          lda scalen,x     Tab. Zeiger in Freq. Konst.
11592  6f25  aa                tax              Zeiger in Tab. Freq. Konst.
11593  6f26
11594  6f26  ===> Rückwärtszähler OCTAVE einrichten <===
11595  6f26
11596  6f26  a9 06             lda #6
11597  6f28  38                sec
11598  6f29  ed 2b 12          sbc octave       OCTAVE Nummer 0-6
11599  6f2c  a8                tay              OCTAVE Rückw. Zähler
11600  6f2d
11601  6f2d  8a                txa              Zeiger in Tab. Freq.Konst.
11602  6f2e  18                clc
11603  6f2f  6d 2c 12          adc sharp        Flag +/- Halbton
11604  6f32  10 03             bpl plnot1
11605  6f34
11606  6f34  ===> Note A um Halbton erniedrigen <===
11607  6f34
11608  6f34  a9 0b             lda #11          Zeiger in Tab. Freq.Konst.
11609  6f36  c8                iny              OCTAVE Rückw. Zähler
11610  6f37
```

```
11611  6f37  c9 0c      plnot1 cmp #12
11612  6f39  90 03             bcc plnot2       Freq. Konst. aus Tabelle
11613  6f3b
11614  6f3b
11614  6f3b  ===> Note G um Halbton erhöhen <===
11615  6f3b
11616  6f3b  a9 00             lda #0           Zeiger in Tab. Freq.Konst.
11617  6f3d  88                dey              OCTAVE Rückw. Zähler
11618  6f3e
11619  6f3e
11620  6f3e  ===> Freq. Konst. aus Tabelle <===
11621  6f3e
11622  6f3e  aa         plnot2 tax              Zeiger in Tab. Freq.Konst.
11623  6f3f
11624  6f3f  bd f9 6f          lda scalel,x     Tab. Low-Bytes Freq.Konst.
11625  6f42  8d 2d 12          sta pitch        Low-Byte Freq.Konst.
11626  6f45
11627  6f45  bd 05 70          lda scaleh,x     Tab. High-Bytes Freq.Konst.
11628  6f48
11629  6f48
11630  6f48  ===> Freq.Konst. für OCTAVE berechnen <===
11631  6f48
11632  6f48  88         plnot3 dey              OCTAVE Rückw. Zähler
11633  6f49  30 27             bmi plnot4       Parameter Note berechen, an SID
11634  6f4b
11635  6f4b  4a                lsr a            High-Byte Freq.Konst.
11636  6f4c  6e 2d 12          ror pitch        Low-Byte Freq.Konst.
11637  6f4f
11638  6f4f  4c 48 6f          jmp plnot3       Freq.Konst. für OCTAVE berechnen
11639  6f52
11640  6f52
11641  6f52  ===> PLAY Befehl bearbeiten <===
11642  6f52
11643  6f52  c9 4d      plcmd  cmp #'m'
11644  6f54  f0 07             beq playm        PLAY Befehl M bearbeiten
11645  6f56
11646  6f56  ===> Flag Übergabe Wert V, O, T, X, U setzen <===
11647  6f56
11648  6f56  bd 1c 9d          lda rbits,x      Tab. Bits7-0
11649  6f59  8d 26 01          sta plyflg       Flag PLAY Befehle
11650  6f5c  60                rts
11651  6f5d
11652  6f5d
11653  6f5d  ===> PLAY Befehl M bearbeiten <===
11654  6f5d
11655  6f5d  a0 05      playm  ldy #5           Zeiger auf High-Bytes
11656  6f5f
11657  6f5f
11658  6f5f  ===> Unterlauf alle Zähler Tondauer abw. <===
11659  6f5f
11660  6f5f  b9 23 12   playml lda voiclo,y     Tab. High-Bytes Tondauer
11661  6f62  10 fb             bpl playml       Unterlauf alle Zähler Tondauer abw.
11662  6f64
11663  6f64  88                dey
11664  6f65  88                dey              Zeiger auf nächstes High-Byte
11665  6f66
11666  6f66  10 f7             bpl playml       Unterlauf alle Zähler Tondauer abw.
11667  6f68
```

```
11668  6f68  60              rts
11669  6f69
11670  6f69
11671  6f69
11671  6f69  ===> Erhöhung um Halbton <===
11672  6f69
11673  6f69  a9 01     plshrp lda #1
11674  6f6b  2c               .byte _skip2
11675  6f6c
11676  6f6c
11677  6f6c  ===> Erniedrigung um Halbton <===
11678  6f6c
11679  6f6c  a9 ff     plflat lda #$ff
11680  6f6e  8d 2c 12         sta sharp        Flag +/- Halbton
11681  6f71
11682  6f71  60               rts
11683  6f72
11684  6f72
11685  6f72  ===> Parameter Note berechnen, an SID <===
11686  6f72
11687  6f72  8d 2e 12  plnot4 sta pitch+1      High-Byte Freq.Konst.
11688  6f75
11689  6f75  a9 00            lda #0           Flag für NOTE
11690  6f77  2c               .byte _skip2
11691  6f78
11692  6f78
11693  6f78  ===> Pause <===
11694  6f78
11695  6f78  a9 ff     plrest lda #$ff         Flag für PAUSE
11696  6f7a  48               pha              Flag NOTE/PAUSE
11697  6f7b
11698  6f7b  ae 2f 12         ldx voice        VOICE Nummer -1
11699  6f7e  bc e4 6f         ldy times2,x     Tab. Zeigerwerte für Voice 1-3
11700  6f81
11701  6f81
11702  6f81  ===> Unterlauf Tondauer akt. VOICE abw. <===
11703  6f81
11704  6f81  b9 24 12  plnlp1 lda voichi,y     Tab. High-Bytes Tondauer (1-3)
11705  6f84  10 fb            bpl plnlp1       Unterlauf Tondauer akt. VOICE abw.
11706  6f86
11707  6f86  ad 29 12         lda ntime        Low-Byte Tondauer Konst.
11708  6f89  99 23 12         sta voiclo,y     Tab. Low-Bytes Tondauer (1-3)
11709  6f8c
11710  6f8c  ad 2a 12         lda ntime+1      High-Byte Tondauer Konst.
11711  6f8f  99 24 12         sta voichi,y     Tab. High-Bytes Tondauer (1-3)
11712  6f92
11713  6f92  ad 33 12         lda dnote        Flag für punktierte Note
11714  6f95
11715  6f95   00101110  punktierte Note (ASCII Punkt)
11716  6f95   00000000  nicht punktierte Note
11717  6f95
11718  6f95  f0 17            beq plnot5
11719  6f97
11720  6f97  ===> Punktierte Note, Freq.Konst. halbieren <===
11721  6f97
11722  6f97  ad 2a 12         lda ntime+1      High-Byte Tondauer Konst.
11723  6f9a  4a               lsr a
11724  6f9b  48               pha
```

```
11725  6f9c
11726  6f9c  ad 29 12          lda ntime        Low-Byte Tondauer Konst.
11727  6f9f  6a                ror a
11728  6fa0  18                clc
11729  6fa1  79 23 12          adc voiclo,y     Tab. Low-Bytes Tondauer (1-3)
11730  6fa4  99 23 12          sta voiclo,y     Tab. Low-Bytes Tondauer (1-3)
11731  6fa7
11732  6fa7  68                pla
11733  6fa8  79 24 12          adc voichi,y     Tab. High-Bytes Tondauer (1-3)
11734  6fab  99 24 12          sta voichi,y     Tab. High-Bytes Tondauer (1-3)
11735  6fae
11736  6fae  68          plnot5 pla              Flag PAUSE/NOTE
11737  6faf
11738  6faf   11111111  PAUSE
11739  6faf   00000000  NOTE
11740  6faf
11741  6faf  30 1d              bmi clrplf       Flags für +/- Halbton löschen
11742  6fb1
11743  6fb1  ===> Parameter Note an SID <===
11744  6fb1
11745  6fb1  20 45 a8          jsr mapio        I/O-Chips einblenden
11746  6fb4
11747  6fb4  bc 39 70          ldy offtab,x     Tab. SID Offsets VOICE 1-3
11748  6fb7  ad 2d 12          lda pitch        Low-Byte Freq.Konst.
11749  6fba  99 00 d4          sta sid+freqlo,y SID Low-Byte Osz.Frequ.
11750  6fbd
11751  6fbd  ad 2e 12          lda pitch+1      High-Byte Freq.Konst.
11752  6fc0  99 01 d4          sta sid+freqhi,y SID High-Byte Osz.Frequ.
11753  6fc3
11754  6fc3  a9 08             lda #%00001000
11755  6fc5
11756  6fc5   ....3...  1 = VOICE Oscillator aus
11757  6fc5
11758  6fc5  99 04 d4          sta sid+osc1+oscctl,y  SID Stimme 1 WELLENFORM
11759  6fc8
11760  6fc8  bd 30 12          lda wave,x       Osc. Control Byte
11761  6fcb  99 04 d4          sta sid+osc1+oscctl,y  SID Stimme 1 WELLENFORM
11762  6fce
11763  6fce
11764  6fce  ===> Flags für +/- Halbton löschen <===
11765  6fce
11766  6fce  a9 00       clrplf lda #0
11767  6fd0  8d 2c 12          sta sharp        Flag +/- Halbton
11768  6fd3  8d 33 12          sta dnote        Flag für punktierte Note
11769  6fd6  60                rts
11770  6fd7
11771  6fd7
11772  6fd7  ===> BASIC-Statement TEMPO ($fe $05) <===
11773  6fd7
11774  6fd7  20 f4 87    tempo  jsr getbyt      8 Bit INT üb. frmnum nach xr
11775  6fda
11776  6fda  Spiel Tempo n
11777  6fda  Spiel Dauer = 19.22/n in Sekunden
11778  6fda
11779  6fda  8a                txa              Spiel Tempo
11780  6fdb  f0 04             beq temper       Behandlung »?illegal quantity error«
11781  6fdd
11782  6fdd  8e 22 12          stx tmprat       Spiel Tempo
```

```
11783  6fe0  60              rts
11784  6fe1
11785  6fe1
11786  6fe1
11786  6fe1  ===> Behandlung »?illegal quantity error« <===
11787  6fe1
11788  6fe1  4c 28 7d   temper jmp fcerr       Behandlung »?illegal quantity error«
11789  6fe4
11790  6fe4
11791  6fe4  ===> Tab. Zeigerwerte für Voice 1-3 <===
11792  6fe4
11793  6fe4  00 02 04   times2 .byte 0, 2, 4
11794  6fe7
11795  6fe7
11796  6fe7  ===> Tab. Befehle Tondauer <===
11797  6fe7
11798  6fe7  57         notes  .byte 'w'       ganze Note
11799  6fe8  48                .byte 'h'       halbe Note
11800  6fe9  51                .byte 'q'       viertel Note
11801  6fea  49                .byte 'i'       achtel Note
11802  6feb  53                .byte 's'       sechzehntel Note
11803  6fec
11804  6fec
11805  6fec  ===> Tab. Befehle PLAY <===
11806  6fec
11807  6fec  56         mutabl .byte 'v'       VOICE (1-3) setzen
11808  6fed  4f                .byte 'o'       OCTAVE (0-6) setzen
11809  6fee  54                .byte 't'       Klanghüllkurve (0-9) setzen
11810  6fef  58                .byte 'x'       Filter schalten (0/1)
11811  6ff0  55                .byte 'u'       Lautstärke (0-9) setzen
11812  6ff1  4d                .byte 'm'       Warten auf Ende Musik
11813  6ff2
11814  6ff2
11815  6ff2  ===> Tab. Zeiger in Freq. Konst <===
11816  6ff2
11817  6ff2  09         scalen .byte 9         Note A
11818  6ff3  0b                .byte 11        Note B
11819  6ff4  00                .byte 0         Note C
11820  6ff5  02                .byte 2         Note D
11821  6ff6  04                .byte 4         Note E
11822  6ff7  05                .byte 5         Note F
11823  6ff8  07                .byte 7         Note G
11824  6ff9
11825  6ff9
11826  6ff9  ===> Tab. Low-Bytes Freq.Konst. <===
11827  6ff9
11828  6ff9  2f         scalel .byte <19503
11829  6ffa  b6                .byte <20662
11830  6ffb  83                .byte <21891
11831  6ffc  99                .byte <23193
11832  6ffd  fc                .byte <24572
11833  6ffe  b1                .byte <26033
11834  6fff  bd                .byte <27581
11835  7000  25                .byte <29221
11836  7001  ef                .byte <30959
11837  7002  20                .byte <32800
11838  7003  be                .byte <34750
11839  7004  d1                .byte <36817
```

```
11840  7005
11841  7005
11842  7005
11842  7005  ===> Tab. High-Bytes Freq.Konst. <===
11843  7005
11844  7005  4c          scaleh .byte >19503
11845  7006  50                 .byte >20662
11846  7007  55                 .byte >21891
11847  7008  5a                 .byte >23193
11848  7009  5f                 .byte >24572
11849  700a  65                 .byte >26033
11850  700b  6b                 .byte >27581
11851  700c  72                 .byte >29221
11852  700d  78                 .byte >30959
11853  700e  80                 .byte >32800
11854  700f  87                 .byte >34750
11855  7010  8f                 .byte >36817
11856  7011
11857  7011
11858  7011  ===> Tab. ATTACK/DECAY HÜLLK. 0-9 <===
11859  7011
11860  7011  09          atkmus .byte %00001001  Klavier
11861  7012  c0                 .byte %11000000  Akkordeon
11862  7013  00                 .byte %00000000  Zirkusorgel
11863  7014  05                 .byte %00000101  Trommel
11864  7015  94                 .byte %10010100  Flöte
11865  7016  09                 .byte %00001001  Gitarre
11866  7017  09                 .byte %00001001  Cembalo
11867  7018  09                 .byte %00001001  Orgel
11868  7019  89                 .byte %10001001  Trompete
11869  701a  09                 .byte %00001001  Xylophon
11870  701b
11871  701b
11872  701b  ===> Tab. SUSTAIN/RELEASE HÜLLK. 0-9 <===
11873  701b
11874  701b  00          susmus .byte %00000000  Klavier
11875  701c  c0                 .byte %11000000  Akkordeon
11876  701d  f0                 .byte %11110000  Zirkusorgel
11877  701e  50                 .byte %01010000  Trommel
11878  701f  40                 .byte %01000000  Flöte
11879  7020  21                 .byte %00100001  Gitarre
11880  7021  00                 .byte %00000000  Cembalo
11881  7022  90                 .byte %10010000  Orgel
11882  7023  41                 .byte %01000001  Trompete
11883  7024  00                 .byte %00000000  Xylophon
11884  7025
11885  7025
11886  7025  ===> Tab. WELLENFORM HÜLLK. 0-9 <===
11887  7025
11888  7025  41          wavmus .byte %01000001  Klavier
11889  7026  21                 .byte %00100001  Akkordeon
11890  7027  11                 .byte %00010001  Zirkusorgel
11891  7028  81                 .byte %10000001  Trommel
11892  7029  11                 .byte %00010001  Flöte
11893  702a  21                 .byte %00100001  Gitarre
11894  702b  41                 .byte %01000001  Cembalo
11895  702c  41                 .byte %01000001  Orgel
11896  702d  41                 .byte %01000001  Trompete
```

```
11897  702e  11               .byte %00010001  Xylophon
11898  702f
11899  702f
11900  702f
11900  702f  ===> Tab. PULSBREITE HÜLLK. 0-9 <===
11901  702f
11902  702f  06         pwhmus .byte %00000110  Klavier
11903  7030  00                .byte %00000000  Akkordeon
11904  7031  00                .byte %00000000  Zirkusorgel
11905  7032  00                .byte %00000000  Trommel
11906  7033  00                .byte %00000000  Flöte
11907  7034  00                .byte %00000000  Gitarre
11908  7035  02                .byte %00000010  Cembalo
11909  7036  08                .byte %00001000  Orgel
11910  7037  02                .byte %00000010  Trompete
11911  7038  00                .byte %00000000  Xylophon
11912  7039
11913  7039
11914  7039  ===> Tab. SID Offsets VOICE 1-3 <===
11915  7039
11916  7039  00 07 0e   offtab .byte osc1, osc2, osc3
11917  703c
11918  703c
11919  703c  ===> Tab. Werte VOLUME 0-9 <===
11920  703c
11921  703c  00 01 03   voltab .byte 0, 1, 3, 5, 7
11921  703f  05 07
11922  7041  08 0a 0c          .byte 8, 10, 12, 14, 15
11922  7044  0e 0f
11923  7046
11924  7046
11925  7046                   .end
11926  7046                   .lib basic.sound_2
11927  7046  put"@basic.sound_2"
```

```
11929  7046
11930  7046
11931  7046  ===> BASIC-Statement FILTER ($fe $03) <===
11932  7046
11933  7046  48         filter pha              nächstes Zeichen
11934  7047
11935  7047  a0 03             ldy #3
11936  7049  b9 71 12   filt10 lda flters,y     Kopie SID Regs. 21-24
11937  704c  99 34 12         sta fltsav,y     2. Kopie SID Regs. 21-24
11938  704f  88               dey
11939  7050  10 f7            bpl filt10
11940  7052
11941  7052  ===> Test Angabe Filter Grenzfrequenz <===
11942  7052
11943  7052  68               pla              nächstes Zeichen
11944  7053  c9 2c            cmp #','
11945  7055  f0 19            beq filt20       Paß Charakteristik einlesen
11946  7057
11947  7057  ===> Filter Grenzfrequenz einlesen <===
11948  7057
11949  7057  20 12 88         jsr getwrd       16 Bit INT nach linnum
11950  705a
11951  705a   Grenzfrequenz 0-2047
11952  705a
11953  705a  c9 08            cmp #>2047+1
11954  705c  b0 60            bcs flterr       Behandlung »?illegal quantity error«
11955  705e
11956  705e  8c 34 12         sty fltsav       2. Kopie SID Reg. 21 (FCLOW)
11957  7061  8c 35 12         sty fltsav+1     2. Kopie SID Reg. 22 (FCHI)
11958  7064
11959  7064  4a               lsr a
11960  7065  6e 35 12         ror fltsav+1     2. Kopie SID Reg. 22 (FCHI)
11961  7068
11962  7068  4a               lsr a
11963  7069  6e 35 12         ror fltsav+1     2. Kopie SID Reg. 22 (FCHI)
11964  706c
11965  706c  4a               lsr a
11966  706d  6e 35 12         ror fltsav+1     Frequenz geteilt durch 8
11967  7070
11968  7070
11969  7070  ===> Paß Charakteristik einlesen <===
11970  7070
11971  7070  a9 10      filt20 lda #%00010000   Initialisierung Tiefpaß
11972  7072
11973  7072   .6.....  1 = Hochpaß
11974  7072   ..5.....  1 = Bandpaß
11975  7072   ...4....  1 = Tiefpaß
11976  7072
11977  7072  8d 38 12         sta fltflg       Bit Muster für Paß Charakteristik
11978  7075
11979  7075
11980  7075  ===> Test auf Angabe Paß ein-/aus <===
11981  7075
11982  7075  20 1e 9e   filt30 jsr optbyt      opt. 8 Bit INT nach xr
11983  7078
11984  7078   0 = Paß aus, 1 = Paß ein
11985  7078
11986  7078  90 17            bcc filt60       keine Angabe
```

```
11987  707a
11988  707a  ===> Paß ein-/ausschalten <===
11989  707a
11990  707a  e0 01             cpx #1
11991  707c  90 05             bcc filt40       Paß aus
11992  707e  f0 03             beq filt40       Paß ein
11993  7080
11994  7080  ===> Behandlung »?illegal quantity error« <===
11995  7080
11996  7080  4c 28 7d          jmp fcerr        Behandlung »?illegal quantity error«
11997  7083
11998  7083
11999  7083  ad 37 12   filt40 lda fltsav+3     2. Kopie SID Reg. 24 (VOL/FILT)
12000  7086  0d 38 12          ora fltflg       Bit Muster für Paß Charakteristik
12001  7089  b0 03             bcs filt50       eingeschaltet
12002  708b
12003  708b  ===> Paß einschalten <===
12004  708b
12005  708b  4d 38 12          eor fltflg       Bit Muster für Paß Charakteristik
12006  708e
12007  708e  8d 37 12   filt50 sta fltsav+3     2. Kopie SID Reg. 24 (VOL/FILT)
12008  7091
12009  7091
12010  7091  ===> Abfrage nächste Paß Charakteristik <===
12011  7091
12012  7091  0e 38 12   filt60 asl fltflg       Bit Muster für Paß Charakteristik
12013  7094  10 df             bpl filt30       Test auf Angabe Paß ein-/aus
12014  7096
12015  7096  ===> Test / Resonanz Wert einlesen <===
12016  7096
12017  7096  20 1e 9e          jsr optbyt       opt. 8 Bit INT nach xr
12018  7099  90 17             bcc filt70       keine Angabe
12019  709b
12020  709b  ===> Resonanz Wert bearbeiten <===
12021  709b
12022  709b  e0 10             cpx #16
12023  709d  b0 1f             bcs flterr       Behandlung »?illegal quantity error«
12024  709f
12025  709f  8a                txa              Resonanz Wert
12026  70a0  0a                asl a
12027  70a1  0a                asl a
12028  70a2  0a                asl a
12029  70a3  0a                asl a
12030  70a4
12031  70a4   7654....  Filter Resonanz Wert
12032  70a4
12033  70a4  8d 39 12          sta nibble       Halb Byte Resonanz Wert
12034  70a7
12035  70a7  ad 36 12          lda fltsav+2     2. Kopie SID Reg. 23 (RESNCE)
12036  70aa
12037  70aa   7654....  Filter Resonanz Wert
12038  70aa   ....3...  Filter EXTERN,  1=ein
12039  70aa   .....2..  Filter VOICE 3, 1=ein
12040  70aa   ......1.  Filter VOICE 2, 1=ein
12041  70aa   .......0  Filter VOICE 1, 1=ein
12042  70aa
12043  70aa  29 0f             and #%00001111
12044  70ac
```

```
12045  70ac  0d 39 12          ora nibble       Halb Byte Resonanz Wert
12046  70af
12047  70af   7654....  Filter Resonanz Wert
12048  70af
12049  70af  8d 36 12          sta fltsav+2     2. Kopie SID Reg. 23 (RESNCE)
12050  70b2
12051  70b2  a0 03       filt70 ldy #3
12052  70b4  b9 34 12    filt80 lda fltsav,y     2. Kopie SID Regs. 21-24
12053  70b7  99 71 12          sta flters,y     Kopie SID Regs. 21-24
12054  70ba  88                dey
12055  70bb  10 f7             bpl filt80
12056  70bd
12057  70bd  60                rts
12058  70be
12059  70be
12060  70be  ===> Behandlung »?illegal quantity error« <===
12061  70be
12062  70be  4c 28 7d    flterr jmp fcerr        Behandlung »?illegal quantity error«
12063  70c1
12064  70c1
12065  70c1  ===> BASIC-Statement ENVELOPE ($fe $0a) <===
12066  70c1
12067  70c1  20 f4 87    envlop jsr getbyt       8 Bit INT üb. frmnum nach xr
12068  70c4
12069  70c4  HÜLLKURVE Nummer 0-9
12070  70c4
12071  70c4  e0 0a             cpx #10
12072  70c6  90 03             bcc tone10       Vorgabe Werte HÜLLKURVE 0-9
12073  70c8
12074  70c8  ===> Behandlung »?illegal quantity error« <===
12075  70c8
12076  70c8  4c 28 7d    tonerr jmp fcerr        Behandlung »?illegal quantity error«
12077  70cb
12078  70cb
12079  70cb  ===> Vorgabe Werte HÜLLKURVE 0-9 <===
12080  70cb
12081  70cb  8e 3a 12    tone10 stx tonnum       Nummer HÜLLKURVE 0-9
12082  70ce
12083  70ce  bd 3f 12          lda atktab,x     Tab. ATTACK/DECAY akt. HÜLLKURVEN
12084  70d1  8d 3b 12          sta tonval       ATTACK/DECAY akt. HÜLLKURVE
12085  70d4
12086  70d4  bd 49 12          lda sustab,x     Tab. SUSTAIN/RELEASE akt. HÜLLK.
12087  70d7  8d 3c 12          sta tonval+1     SUSTAIN/RELEASE akt. HÜLLKURVE
12088  70da
12089  70da  bd 53 12          lda wavtab,x     Tab. WELLENFORMEN akt. HÜLLKURVEN
12090  70dd  8d 3d 12          sta tonval+2     WELLENFORM akt. HÜLLKURVE
12091  70e0
12092  70e0  ===> Test / Definition HÜLLKURVE 0-9 <===
12093  70e0
12094  70e0  a2 00             ldx #0
12095  70e2  8e 3e 12    tone20 stx parcnt       Zeiger in Param. HÜLLKURVE
12096  70e5
12097  70e5   ATTACK  (Anschlag Zeit)   0-15
12098  70e5   DECAY   (Abschwell Zeit)  0-15
12099  70e5   SUSTAIN (Halte Zeit)      0-15
12100  70e5   RELEASE (Auskling Zeit)   0-15
12101  70e5
12102  70e5   WELLENFORM                0 = Dreieck
```

```
12103  70e5                              1 = Sägezahn
12104  70e5                              2 = Rechteck
12105  70e5                              3 = Rauschen
12106  70e5                              4 = Ringmodulation
12107  70e5
12108  70e5   PULSBREITE                 0-4095
12109  70e5
12110  70e5  ===> Test / Parameter einlesen <===
12111  70e5
12112  70e5  20 1e 9e          jsr optbyt       opt. 8 Bit INT nach xr
12113  70e8  90 16             bcc tone30       keine Angabe
12114  70ea
12115  70ea  ===> High-Halb Byte eingelesen <===
12116  70ea
12117  70ea  8a                txa
12118  70eb  0a                asl a
12119  70ec  0a                asl a
12120  70ed  0a                asl a
12121  70ee  0a                asl a
12122  70ef  8d 39 12          sta nibble       High Nibble
12123  70f2
12124  70f2  ===> High-Halbbyte einblenden <===
12125  70f2
12126  70f2  ae 3e 12          ldx parcnt
12127  70f5  bd 3b 12          lda tonval,x     ATTACK/DECAY akt. HÜLLKURVE
12128  70f8  29 0f             and #%0001111
12129  70fa  0d 39 12          ora nibble       Puffer oberes/unteres Halb Byte
12130  70fd  9d 3b 12          sta tonval,x     ATTACK/DECAY akt. HÜLLKURVE
12131  7100
12132  7100
12133  7100  ===> Test / Parameter einlesen <===
12134  7100
12135  7100  20 1e 9e   tone30 jsr optbyt       opt. 8 Bit INT nach xr
12136  7103  90 14             bcc tone40       keine Angabe
12137  7105
12138  7105  ===> Low-Halb Byte eingelesen <===
12139  7105
12140  7105  8a                txa
12141  7106  29 0f             and #%00001111
12142  7108  8d 39 12          sta nibble       Low Nibble
12143  710b
12144  710b  ===> Low-Halbbyte einblenden <===
12145  710b
12146  710b  ae 3e 12          ldx parcnt
12147  710e  bd 3b 12          lda tonval,x     ATTACK/DECAY akt. HÜLLKURVE
12148  7111  29 f0             and #%11110000
12149  7113  0d 39 12          ora nibble       Puffer oberes/unteres Halb Byte
12150  7116  9d 3b 12          sta tonval,x     ATTACK/DECAY akt. HÜLLKURVE
12151  7119
12152  7119  ae 3e 12   tone40 ldx parcnt       Zeiger in Param. HÜLLKURVE
12153  711c  e8                inx
12154  711d
12155  711d  ===> Test / max. zwei Durchgänge <===
12156  711d
12157  711d  e0 01             cpx #1
12158  711f  f0 c1             beq tone20
12159  7121
```

```
12160  7121  ===> Wert für WELLENFORM einlesen <===
12161  7121
12162  7121  20 1e 9e          jsr optbyt       opt. 8 Bit INT nach xr
12163  7124  90 10             bcc tone90       keine Angabe
12164  7126
12165  7126  ===> WELLENFORM bearbeiten <===
12166  7126
12167  7126  a9 15             lda #%00010101   Dreieck, Ringmod., Start
12168  7128
12169  7128   ...4....  1 = Dreieckschwingung ein
12170  7128   .....2..  1 = Ringmodulation ein
12171  7128   .......0  1 = Start ATTACK/DECAY/SUSTAIN
12172  7128
12173  7128  e0 04             cpx #4           Ringmodulation
12174  712a  f0 07             beq tone80
12175  712c  b0 9a             bcs tonerr       Behandlung »?illegal quantity error«
12176  712e
12177  712e  bd b7 6c          lda sbits+4,x    Tabelle Maskenbits Bit0-7
12178  7131
12179  7131   7.......  1 = Rauschgenerator ein
12180  7131   .6......  1 = Rechteckschwingung ein
12181  7131   ..5.....  1 = Sägezahnschwingung ein
12182  7131   ...4....  1 = Dreieckschwingung ein
12183  7131
12184  7131  09 01             ora #%00000001
12185  7133
12186  7133   .......0  1 = Start ATTACK/DECAY/SUSTAIN
12187  7133
12188  7133  8d 3d 12   tone80 sta tonval+2     WELLENFORM akt. HÜLLKURVE
12189  7136
12190  7136
12191  7136  ===> Test / PULSBREITE einlesen <===
12192  7136
12193  7136  20 06 9e   tone90 jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
12194  7139  90 13             bcc ton110       keine Angabe
12195  713b
12196  713b  ===> Bearbeitung PULSBREITE <===
12197  713b
12198  713b  aa                tax              High-Byte PULSBREITE
12199  713c
12200  713c  ad 3d 12          lda tonval+2     WELLENFORM akt. HÜLLKURVE
12201  713f  29 40             and #%01000000
12202  7141
12203  7141   .6......  1 = Rechteckschwingung ein
12204  7141
12205  7141  f0 0b             beq ton110
12206  7143
12207  7143  ===> PULSBREITE Rechteckschwingung eintragen <===
12208  7143
12209  7143  8a                txa              High-Byte PULSBREITE
12210  7144
12211  7144  ae 3a 12          ldx tonnum       Nummer HÜLLKURVE 0-9
12212  7147  9d 67 12          sta pulshi,x     Tab. High-Byte PULSBREITE akt. HÜLLK.
12213  714a
12214  714a  98                tya              Low-Byte PULSBREITE
12215  714b  9d 5d 12          sta pulslo,x     Tab. Low-Bytes PULSBREITE akt. HÜLLK.
12216  714e
12217  714e  ae 3a 12   ton110 ldx tonnum       Nummer HÜLLKURVE 0-9
```

```
12218  7151  ad 3b 12          lda tonval       ATTACK/DECAY akt. HÜLLKURVE
12219  7154  9d 3f 12          sta atktab,x     Tab. ATTACK/DECAY akt. HÜLLKURVEN
12220  7157
12221  7157  ad 3c 12          lda tonval+1     SUSTAIN/RELEASE akt. HÜLLKURVE
12222  715a  9d 49 12          sta sustab,x     Tab. SUSTAIN/RELEASE akt. HÜLLK.
12223  715d
12224  715d  ad 3d 12          lda tonval+2     WELLENFORM akt. HÜLLKURVE
12225  7160  9d 53 12          sta wavtab,x     Tab. WELLENFORMEN akt. HÜLLKURVEN
12226  7163
12227  7163  60                rts
12228  7164
12229  7164
12230  7164  ===> BASIC-Statement COLLISION ($fe $17) <===
12231  7164
12232  7164  20 f4 87   colisn jsr getbyt      8 Bit INT üb. frmnum nach xr
12233  7167
12234  7167
12235  7167   1 = Sprite Sprite Kollision
12236  7167   2 = Sprite Hintergrund Kollision
12237  7167   3 = Light Pen Aktivierung
12238  7167
12239  7167  ca                dex
12240  7168  e0 03             cpx #3
12241  716a  b0 21             bcs clsner       Behandlung »?illegal quantity error«
12242  716c
12243  716c  8e 80 12          stx coltyp       COLLISION Typ (0-2)
12244  716f
12245  716f  ===> Test / Zeilennummer einlesen <===
12246  716f
12247  716f  20 06 9e          jsr optwrd       opt. 16 Bit INT nach yr/ac, sonst 0
12248  7172  08                php
12249  7173
12250  7173  ae 80 12          ldx coltyp       COLLISION Typ (0-2)
12251  7176  9d 7c 12          sta intadh,x     Tab. High-Byte Zeilennr. COLISN
12252  7179
12253  7179  98                tya              Low-Byte Zeilennummer
12254  717a  9d 79 12          sta intadl,x     Tab. Low-Byte Zeilennr. COLISN
12255  717d
12256  717d  ad 7f 12          lda intval       Flag für Bearbeitung COLLISION
12257  7180  1d b3 6c          ora sbits,x      Tabelle Maskenbits Bit0-7
12258  7183
12259  7183  28                plp
12260  7184  b0 03             bcs clsn10       Angabe Zeilennummer
12261  7186
12262  7186  ===> Bit Kollisions Typ löschen <===
12263  7186
12264  7186  5d b3 6c          eor sbits,x      Tabelle Maskenbits Bit0-7
12265  7189
12266  7189  8d 7f 12   clsn10 sta intval       Flag für Bearbeitung COLLISION
12267  718c  60                rts
12268  718d
12269  718d
12270  718d  ===> Behandlung »?illegal quantity error« <===
12271  718d
12272  718d  4c 28 7d   clsner jmp fcerr        Behandlung »?illegal quantity error«
12273  7190
12274  7190
```

```
12275  7190  ===> BASIC-Statement SPRCOLOR ($fe $08) <===
12276  7190
12277  7190  c9 2c      sprcol cmp #','
12278  7192  f0 0e             beq sprcl1       keine Angabe Zusatzfarbe 1
12279  7194
12280  7194
12280  7194  ===> Sprite Zusatzfarbe 1 einlesen <===
12281  7194
12282  7194  20 f4 87          jsr getbyt       8 Bit INT üb. frmnum nach xr
12283  7197  ca                dex
12284  7198  e0 10             cpx #16
12285  719a  b0 17             bcs sprcer       Behandlung »?illegal quantity error«
12286  719c
12287  719c  20 45 a8          jsr mapio        I/O-Chips einblenden
12288  719f  8e 25 d0          stx vicmc0       VIC 8564 Sprite Multicolor #0
12289  71a2
12290  71a2
12291  71a2  ===> Test / Sprite Zusatzfarbe 2 einlesen <===
12292  71a2
12293  71a2  20 1e 9e   sprcl1 jsr optbyt       opt. 8 Bit INT nach xr
12294  71a5  90 0b             bcc sprcl2       keine Angabe
12295  71a7
12296  71a7  ca                dex
12297  71a8  e0 10             cpx #16
12298  71aa  b0 07             bcs sprcer       Behandlung »?illegal quantity error«
12299  71ac
12300  71ac  20 45 a8          jsr mapio        I/O-Chips einblenden
12301  71af  8e 26 d0          stx vicmc1       VIC 8564 Sprite Multicolor #1
12302  71b2
12303  71b2  60         sprcl2 rts
12304  71b3
12305  71b3
12306  71b3  ===> Behandlung »?illegal quantity error« <===
12307  71b3
12308  71b3  4c 28 7d   sprcer jmp fcerr        Behandlung »?illegal quantity error«
12309  71b6
12310  71b6
12311  71b6  ===> BASIC-Statement WIDTH ($fe $1c) <===
12312  71b6
12313  71b6  20 f4 87   setwid jsr getbyt       8 Bit INT üb. frmnum nach xr
12314  71b9
12315  71b9   1 = einfache Strich Stärke
12316  71b9   2 = doppelte Strich Stärke
12317  71b9
12318  71b9  ca                dex
12319  71ba  e0 02             cpx #2
12320  71bc  b0 04             bcs widerr       Behandlung »?illegal quantity error«
12321  71be
12322  71be  8e 6b 11          stx width        Strichstärke (0-1)
12323  71c1  60                rts
12324  71c2
12325  71c2
12326  71c2  ===> Behandlung »?illegal quantity error« <===
12327  71c2
12328  71c2  4c 28 7d   widerr jmp fcerr        Behandlung »?illegal quantity error«
12329  71c5
12330  71c5
```

```
12331  71c5  ===> BASIC-Statement VOL ($db) <===
12332  71c5
12333  71c5  20 f4 87   volume jsr getbyt       8 Bit INT üb. frmnum nach xr
12334  71c8
12335  71c8   Lautstärke 0-15
12336  71c8
12337  71c8  e0 10             cpx #16
12338  71ca  b0 1d             bcs volerr       Behandlung »?illegal quantity error«
12339  71cc
12340  71cc  86 77             stx mltus1       Lautstärke 0-15
12341  71ce
12342  71ce  ad 74 12          lda flters+3     Kopie SID Reg. 24 (VOL/FILT)
12343  71d1
12344  71d1   ....3210  0-15 Lautstärke
12345  71d1
12346  71d1  29 f0             and #%11110000
12347  71d3  05 77             ora mltus1       Lautstärke 0-15
12348  71d5  8d 74 12          sta flters+3     Kopie SID Reg. 24 (VOL/FILT)
12349  71d8
12350  71d8  ad 75 12          lda oldvol       letzter Wert SID VOL/FILT
12351  71db
12352  71db   ....3210  0-15 Lautstärke
12353  71db
12354  71db  29 f0             and #%11110000
12355  71dd  05 77             ora mltus1       Lautstärke 0-15
12356  71df  8d 75 12          sta oldvol       letzter Wert SID VOL/FILT
12357  71e2
12358  71e2  20 45 a8          jsr mapio        I/O-Chips einblenden
12359  71e5  8d 18 d4          sta sid+fltvol   SID Filterart / Lautstärke
12360  71e8  60                rts
12361  71e9
12362  71e9
12363  71e9  ===> Behandlung »?illegal quantity error« <===
12364  71e9
12365  71e9  4c 28 7d   volerr jmp fcerr        Behandlung »?illegal quantity error«
12366  71ec
12367  71ec
12368  71ec  ===> BASIC-Statement SOUND ($da) <===
12369  71ec
12370  71ec  20 f4 87   sound  jsr getbyt       8 Bit INT üb. frmnum nach xr
12371  71ef
12372  71ef   VOICE 1-3
12373  71ef
12374  71ef  ca                dex
12375  71f0  e0 03             cpx #3
12376  71f2  90 03             bcc sound1
12377  71f4
12378  71f4  ===> Behandlung »?illegal quantity error« <===
12379  71f4
12380  71f4  4c 28 7d   snderr jmp fcerr        Behandlung »?illegal quantity error«
12381  71f7
12382  71f7
12383  71f7  8e 81 12   sound1 stx snvoic       VOICE 0-2
12384  71fa
12385  71fa  ===> SOUND Frequenz einlesen <===
12386  71fa
12387  71fa  20 0f 88          jsr comwrd       Kommatest, 16 Bit INT nach linnum
12388  71fd
```

```
12389  71fd  8c a5 12          sty tmmaxl      Low-Byte Grenzfrequenz 1
12390  7200  8d a6 12          sta tmmaxh      High-Byte Grenzfrequenz 1
12391  7203
12392  7203  8c ac 12          sty tmfrql      Low-Byte Frequenz
12393  7206  8d ad 12          sta tmfrqh      High-Byte Frequenz
12394  7209
12395  7209
12395  7209  ===> Tondauer einlesen <===
12396  7209
12397  7209  20 0f 88          jsr comwrd      Kommatest, 16 Bit INT nach linnum
12398  720c
12399  720c   SOUND Tondauer 0-32767
12400  720c
12401  720c  c9 80             cmp #>32767+1
12402  720e  b0 e4             bcs snderr      Behandlung »?illegal quantity error«
12403  7210
12404  7210  8c a3 12          sty tmtiml      Low-Byte Tondauer
12405  7213  8d a4 12          sta tmtimh      High-Byte Tondauer
12406  7216
12407  7216  ===> SOUND Richtung einlesen <===
12408  7216
12409  7216  20 1c 9e          jsr optzer      opt. 8 Bit INT nach xr, sonst 0
12410  7219
12411  7219   SOUND Richtung einlesen
12412  7219
12413  7219   0 = zunehmende Klangstufen
12414  7219   1 = abnehmende Klangstufen
12415  7219   2 = oszillierende Klangstufen
12416  7219
12417  7219  e0 03             cpx #3
12418  721b  b0 d7      sound2 bcs snderr      Behandlung »?illegal quantity error«
12419  721d
12420  721d  8a                txa
12421  721e  8d a9 12          sta tmpdir      Richtung 0-2
12422  7221
12423  7221  29 01             and #%00000001
12424  7223  08                php
12425  7224
12426  7224  ===> Test / Grenzfrequenz einlesen <===
12427  7224
12428  7224  20 06 9e          jsr optwrd      opt. 16 Bit INT nach yr/ac, sonst 0
12429  7227
12430  7227   Grenzfrequenz 0-65535
12431  7227
12432  7227  8c a7 12          sty tmminl      Low-Byte Grenzfrequenz 2
12433  722a  8d a8 12          sta tmminh      High-Byte Grenzfrequenz 2
12434  722d
12435  722d  ===> Test / Schrittweite einlesen <===
12436  722d
12437  722d  20 06 9e          jsr optwrd      opt. 16 Bit INT nach yr/ac, sonst 0
12438  7230
12439  7230  28                plp
12440  7231  f0 0d             beq sound3      Richtung 0 oder 2
12441  7233
12442  7233  ===> Schrittweite abnehmende Klangstufen <===
12443  7233
12444  7233  48                pha             High-Byte Schrittweite
12445  7234
```

```
12446  7234  98              tya             Low-Byte Schrittweite
12447  7235  49 ff           eor #%11111111
12448  7237  18              clc
12449  7238  69 01           adc #%00000001
12450  723a  a8              tay             Low-Byte neg. Schrittweite
12451  723b
12452  723b  68              pla             High-Byte Schrittweite
12453  723c  49 ff           eor #%11111111
12454  723e  69 00           adc #%00000000
12455  7240  8d ab 12  sound3 sta tmstph      High-Byte Schrittweite
12456  7243
12457  7243  98              tya
12458  7244  8d aa 12        sta tmstpl      Low-Byte Schrittweite
12459  7247
12460  7247  ===> Test / WELLENFORM einlesen <===
12461  7247
12462  7247  a2 02           ldx #2          Vorgabe Rechteckschwingung
12463  7249  20 1e 9e        jsr optbyt      opt. 8 Bit INT nach xr
12464  724c
12465  724c   WELLENFORM
12466  724c
12467  724c   0 = Dreieck
12468  724c   1 = Sägezahn
12469  724c   2 = Rechteck
12470  724c   3 = Rauschen
12471  724c
12472  724c  e0 04           cpx #4
12473  724e  b0 cb           bcs sound2      Behandlung »?illegal quantity error«
12474  7250
12475  7250  bd b7 6c        lda sbits+4,x   Tabelle Maskenbits Bit0-7
12476  7253
12477  7253   7.......  1 = Rauschgenerator ein
12478  7253   .6......  1 = Rechteckschwingung ein
12479  7253   ..5.....  1 = Sägezahnschwingung ein
12480  7253   ...4....  1 = Dreieckschwingung ein
12481  7253
12482  7253  09 01           ora #%00000001  Oscillator Gate-Bit setzen
12483  7255
12484  7255   .......0  1 = Start ATTACK/DECAY/SUSTAIN
12485  7255
12486  7255  8d b0 12        sta tmwave      WELLENFORM
12487  7258
12488  7258  ===> Test / PULSBREITE einlesen <===
12489  7258
12490  7258  20 06 9e        jsr optwrd      opt. 16 Bit INT nach yr/ac, sonst 0
12491  725b
12492  725b   PULSBREITE 0-4095
12493  725b
12494  725b  b0 04           bcs sound4
12495  725d
12496  725d  ===> Vorgabewert PULSBREITE (50% Tastverhältnis) <===
12497  725d
12498  725d  a9 08           lda #>2048
12499  725f  a0 00           ldy #<2048
12500  7261
12501  7261  c9 10     sound4 cmp #>4095+1
12502  7263  b0 8f           bcs snderr      Behandlung »?illegal quantity error«
12503  7265
```

```
12504  7265  8c ae 12          sty tmpul        Low-Byte PULSBREITE
12505  7268  8d af 12          sta tmpulh       High-Byte PULSBREITE
12506  726b
12507  726b  ad a3 12          lda tmtiml       Low-Byte Tondauer
12508  726e  0d a4 12          ora tmtimh       High-Byte Tondauer
12509  7271  f0 46             beq sound9       Tondauer = 0
12510  7273
12511  7273  ae 81 12          ldx snvoic       VOICE 0-2
12512  7276  8a                txa
12513  7277
12514  7277  0a                asl a
12515  7278  a8                tay
12516  7279
12517  7279
12518  7279  ===> Warten auf Unterlauf Tondauer <===
12519  7279
12520  7279  b9 24 12    sound5 lda voichi,y     Tab. High-Bytes Tondauer (1-3)
12521  727c  10 fb              bpl sound5       Warten auf Unterlauf Tondauer
12522  727e
12523  727e
12524  727e  ===> Warten auf Unterlauf SOUND Tondauer <===
12525  727e
12526  727e  bd 85 12    sound6 lda sndth1,x     High-Byte Tondauer VOICE 1
12527  7281  10 fb              bpl sound6       Warten auf Unterlauf SOUND Tondauer
12528  7283
12529  7283  a0 00              ldy #0
12530  7285  b9 a5 12    sound7 lda tmmaxl,y     Low-Byte Grenzfrequenz 1
12531  7288  9d 88 12          sta sdmxl1,x     Low-Byte Grenzfreq.1 VOICE 1
12532  728b
12533  728b  e8                inx
12534  728c  e8                inx12535  728d  e8                inx
12536  728e
12537  728e  c8                iny
12538  728f  c0 09             cpy #tmpul-tmmaxl
12539  7291  d0 f2             bne sound7
12540  7293
12541  7293  ae 81 12          ldx snvoic       VOICE 0-2
12542  7296  bc 39 70          ldy offtab,x     Tab. SID Offsets VOICE 1-3
12543  7299  20 45 a8          jsr mapio        I/O-Chips einblenden
12544  729c
12545  729c  a9 08             lda #%00001000
12546  729e
12547  729e   00001000  1 = Oszillator aus12548  729e
12549  729e  99 04 d4          sta sid+osc1+oscctl,y  SID Stimme 1 WELLENFORM
12550  72a1
12551  72a1  a9 00             lda #%00000000
12552  72a3
12553  72a3   7654....  ATTACK 0-15
12554  72a3   ....3210  DECAY  0-15
12555  72a3
12556  72a3  99 05 d4          sta sid+atkdcy,y SID Attack/Decay Zeiten
12557  72a6
12558  72a6  a9 f0             lda #%11110000
12559  72a8
12560  72a8   7654....  SUSTAIN 0-15
12561  72a8   ....3210  RELEASE 0-15
12562  72a8
12563  72a8  99 06 d4          sta sid+susrel,y SID Sustain/Release Zeiten
```

```
12564  72ab
12565  72ab  a2 00             ldx #0
12566  72ad  bd ac 12   sound8 lda tmfrql,x     Low-Byte Frequenz
12567  72b0
12568  72b0   $d400 = FREQLO
12569  72b0   $d401 = FREQHI
12570  72b0   $d402 = PULSEF
12571  72b0   $d403 = PULSEC
12572  72b0   $d404 = OSCCTL
12573  72b0
12574  72b0  99 00 d4          sta sid+freqlo,y SID Low-Byte Osz.Frequ.
12575  72b3
12576  72b3  c8                iny
12577  72b4  e8                inx
12578  72b5  e0 05             cpx #5
12579  72b7  d0 f4             bne sound8
12580  72b9
12581  72b9  ae 81 12   sound9 ldx snvoic       VOICE 0-2
12582  72bc
12583  72bc  ac a3 12          ldy tmtiml       Low-Byte Tondauer
12584  72bf  ad a4 12          lda tmtimh       High-Byte Tondauer
12585  72c2
12586  72c2  78                sei
12587  72c3  9d 85 12          sta sndth1,x     High-Byte Tondauer VOICE 1
12588  72c6
12589  72c6  98                tya
12590  72c7  9d 82 12          sta sndtl1,x     Low-Byte Tondauer VOICE 1
12591  72ca
12592  72ca  58                cli
12593  72cb  60                rts
12594  72cc
12595  72cc
12596  72cc  ===> BASIC-Statement WINDOW ($fe $1a) <===
12597  72cc
12598  72cc  20 f4 87   window jsr getbyt       8 Bit INT üb. frmnum nach xr
12599  72cf
12600  72cf   Spalte BS Fenster li.ob. (0-39 bzw. 0-79)
12601  72cf
12602  72cf  e0 28             cpx #40
12603  72d1
12604  72d1  24 d7             bit mode         BS-Modus 40/80-Zeichen
12605  72d3  10 02             bpl wndw10       40 Zeichen BS
12606  72d5
12607  72d5  e0 50             cpx #80
12608  72d7
12609  72d7  b0 59      wndw10 bcs wnderr       Behandlung »?illegal quantity error«
12610  72d9
12611  72d9  8e b3 12          stx wintmp       WINDOW Spalte BS Fenster li.ob.
12612  72dc
12613  72dc  ===> Zeile BS Fenster li.ob. einlesen <===
12614  72dc
12615  72dc  20 09 88          jsr combyt       Kommatest, 8 Bit INT nach xr
12616  72df
12617  72df   Zeile BS Fenster li.ob. (0-24)
12618  72df
12619  72df  e0 19             cpx #25
12620  72e1  b0 4f             bcs wnderr       Behandlung »?illegal quantity error«
12621  72e3
```

```
12622  72e3  8e b4 12          stx wintmp+1     WINDOW Zeile BS Fenster li.ob.
12623  72e6
12624  72e6  ===> Spalte BS Fenster re.unt. einlesen <===
12625  72e6
12626  72e6  20 09 88          jsr combyt       Kommatest, 8 Bit INT nach xr
12627  72e9
12628  72e9   Spalte BS Fenster re.unt. (0-39 bzw. 0-79)
12629  72e9
12630  72e9  e0 28             cpx #40
12631  72eb
12632  72eb  24 d7             bit mode         BS-Modus 40/80-Zeichen
12633  72ed  10 02             bpl wndw20       40 Zeichen BS
12634  72ef
12635  72ef  e0 50             cpx #80
12636  72f1
12637  72f1  b0 3f       wndw20 bcs wnderr       Behandlung »?illegal quantity error«
12638  72f3
12639  72f3  8e b5 12          stx wintmp+2     WINDOW Spalte BS Fenster re.unt.
12640  72f6  ec b3 12          cpx wintmp       WINDOW Spalte BS Fenster li.ob.
12641  72f9  90 37             bcc wnderr       Behandlung »?illegal quantity error«
12642  72fb
12643  72fb  ===> Zeile BS Fenster re.unt. einlesen <===
12644  72fb
12645  72fb  20 09 88          jsr combyt       Kommatest, 8 Bit INT nach xr
12646  72fe
12647  72fe   Zeile BS Fenster li.ob. (0-24)
12648  72fe
12649  72fe  e0 19             cpx #25
12650  7300  b0 30             bcs wnderr       Behandlung »?illegal quantity error«
12651  7302
12652  7302  8e b6 12          stx wintmp+3     WINDOW Zeile BS Fenster re.unt.
12653  7305  ec b4 12          cpx wintmp+1     WINDOW Zeile BS Fenster li.ob.
12654  7308  90 28             bcc wnderr       Behandlung »?illegal quantity error«
12655  730a
12656  730a  ===> WINDOW Modus (HOME/CLR) einlesen <===
12657  730a
12658  730a  20 1c 9e          jsr optzer       opt. 8 Bit INT nach xr, sonst 0
12659  730d
12660  730d   0 = Modus HOME (Vorgabe)
12661  730d   1 = Modus CLR
12662  730d
12663  730d  e0 02             cpx #2
12664  730f  b0 21             bcs wnderr       Behandlung »?illegal quantity error«
12665  7311
12666  7311  8a                txa
12667  7312  48                pha
12668  7313
12669  7313  ae b3 12          ldx wintmp       WINDOW Spalte BS Fenster li.ob.
12670  7316  ad b4 12          lda wintmp+1     WINDOW Zeile BS Fenster li.ob.
12671  7319  18                clc              Flag für li.ob. BS Fenster Ecke
12672  731a  20 2d c0          jsr jwind        BS Fenster definieren
12673  731d
12674  731d  ae b5 12          ldx wintmp+2     WINDOW Spalte BS Fenster re.unt.
12675  7320  ad b6 12          lda wintmp+3     WINDOW Zeile BS Fenster re.unt.
12676  7323  38                sec              Flag für re.unt. BS Fenster Ecke
12677  7324  20 2d c0          jsr jwind        BS Fenster definieren
12678  7327
12679  7327  a2 13             ldx #_home
```

```
12680  7329
12681  7329  68              pla             WINDOW Modus
12682  732a  f0 02           beq wndw30
12683  732c
12684  732c  a2 93           ldx #_clr
12685  732e  8a       wndw30 txa
12686  732f  4c 69 92         jmp cbsout      ind. Ausgabe ac auf akt. Kanal
12687  7332
12688  7332
12689  7332
12689  7332  ===> Behandlung »?illegal quantity error« <===
12690  7332
12691  7332  4c 28 7d  wnderr jmp fcerr       Behandlung »?illegal quantity error«
12692  7335
12693  7335
12694  7335                   .end
12695  7335                   .lib basic.boot
12696  7335  put"@basic.boot"
```

```
12698  7335
12699  7335
12700  7335  ===> BASIC-Statement BOOT ($fe $1c) <===
12701  7335
12702  7335  a9 e6      boot   lda #255-_d1-_fa-_fn1  erlaubt
12703  7337  a2 fc             ldx #255-_stadr-_bnk   erlaubt
12704  7339  20 c3 a3         jsr dosprx·      Param. einlesen/mit ac/xr prüfen
12705  733c
12706  733c  ===> Test auf Dateinamen <===
12707  733c
12708  733c  a5 80            lda parsts       DOS Syntax Byte 1
12709  733e  4a              lsr a
12710  733f
12711  733f   .......0  1 = 1. Filename gelesen
12712  733f
12713  733f  90 1d           bcc boot20       1. Filename nicht gelesen
12714  7341
12715  7341  20 1f a2        jsr blodnp       BLOAD, Param. eingelesen
12716  7344  b0 29           bcs booter       Behandlung I/O Fehler
12717  7346
12718  7346  ae d5 03        ldx curbnk       akt. Bank
12719  7349
12720  7349  a5 81           lda parstx       DOS Syntax Byte 2
12721  734b  4a              lsr a
12722  734c
12723  734c   .......0  1 = Banknummer gelesen
12724  734c
12725  734c  90 03           bcc boot10       keine Angabe
12726  734e
12727  734e  ae 1f 01        ldx dosbnk       Bank Nummer
12728  7351
12729  7351
12730  7351  ===> BOOT Filename / PGM starten <===
12731  7351
12732  7351  86 02      boot10 stx bnk        Bank Nummer
12733  7353
12734  7353  a5 ac             lda sal          ADDRL für LOAD/SAVE
12735  7355  85 04             sta pclo         Puffer 8502 CPU ADDRL PC
12736  7357
12737  7357  a5 ad             lda sah          ADDRH für LOAD/SAVE
12738  7359  85 03             sta pchi         Puffer 8502 CPU ADDRH PC
12739  735b
12740  735b  4c 6e ff          jmp kjsrfr       JSR in andere Bank, RTS in akt. Bank
                                                (Sprungv.)
12741  735e
12742  735e
12743  735e  ===> BOOT CP/M / starten <===
12744  735e
12745  735e  ad 12 01   boot20 lda dosds1       1. Laufwerk
12746  7361  09 30            ora #'0'         Lw nach ASCII
12747  7363  ae 1c 01        ldx dosfa        Primäradresse
12748  7366
12749  7366  20 45 a8        jsr mapio        I/O-Chips einblenden
12750  7369  20 53 ff        jsr kbotcl       Boot-Programm von Diskette laden
                                                (Sprungv.)
12751  736c
12752  736c  b0 01           bcs booter       Behandlung I/O Fehler
12753  736e
```

```
12754  736e  60                rts
12755  736f
12756  736f
12757  736f  ===> Behandlung BOOT I/O Fehler <===
12758  736f
12759  736f  4c d0 90    booter jmp erexit       Behandlung I/O Fehler
12760  7372
12761  7372
12762  7372                    .end
12763  7372                    .lib basic.sprite_2
12764  7372  put"@basic.sprite_2"
```

```
12766  7372
12767  7372
12768  7372  ===> BASIC-Statement SPRDEF ($fe $1d) <===
12769  7372
12770  7372  20 4f 9f   sprdef jsr sethir       Test / GRAPHIC RAM einrichten
12771  7375  20 45 a8           jsr mapio       I/O-Chips einblenden
12772  7378
12773  7378  a9 d0              lda #>chrgug     ADDRH Char.gen. Groß/Graphik
12774  737a  8d 68 11           sta charom
12775  737d
12776  737d  a9 20              lda #%00100000   HI RES
12777  737f  85 d8              sta graphm       Flag Text/Graphik/Split-BS
12778  7381
12779  7381  20 30 6b           jsr clrgrf       GRAPHIC RAM und Farb RAM löschen
12780  7384
12781  7384  ===> Rahmen um Sprite Raster zeichnen <===
12782  7384
12783  7384  a0 80              ldy #%10000000
12784  7386  8c 3d 11           sty xysgn        RVS Flag
12785  7389
12786  7389  ===> Rahmen vertikal <===
12787  7389
12788  7389  a0 18              ldy #24          BS Spalte
12789  738b  a9 20              lda #_spc
12790  738d  a2 00              ldx #0           BS Zeile
12791  738f
12792  738f  20 db 68   sprd05  jsr chrdsp       Zeichen in Bitspeicher bringen
12793  7392  e8                inx              Zeile inkrementieren
12794  7393
12795  7393  e0 15              cpx #21          Zeile
12796  7395  90 f8              bcc sprd05
12797  7397
12798  7397
12799  7397  ===> Rahmen horizontal <===
12800  7397
12801  7397  20 db 68   sprd10  jsr chrdsp       Zeichen in Bitspeicher bringen
12802  739a  88                dey              Spalte
12803  739b  10 fa              bpl sprd10
12804  739d
12805  739d  20 45 a8           jsr mapio        I/O-Chips einblenden
12806  73a0
12807  73a0  ===> Spriteraster mit '+' in Hintergr.fb. füllen <===
12808  73a0
12809  73a0  a5 f1              lda attrib       akt. Attribut
12810  73a2  48                pha
12811  73a3
12812  73a3  ad 21 d0           lda vicbkg       VIC 8564 Hintergrundfarbe
12813  73a6  85 f1              sta attrib       akt. Attribut
12814  73a8
12815  73a8  a9 2b              lda #'+'
12816  73aa  a2 00              ldx #0           Start Zeile
12817  73ac
12818  73ac  8e 3d 11           stx xysgn        RVS Flag löschen
12819  73af
12820  73af  a0 00      sprd15  ldy #0           Spalte
12821  73b1  20 db 68   sprd20  jsr chrdsp       Zeichen in Bitspeicher bringen
12822  73b4
12823  73b4  c8                iny              Spalte
```

```
12824  73b5  c0 18            cpy #24
12825  73b7  90 f8            bcc sprd20
12826  73b9
12827  73b9  e8               inx             Zeile
12828  73ba  e0 15            cpx #21
12829  73bc  90 f1            bcc sprd15
12830  73be
12831  73be  68               pla             akt. Attribut
12832  73bf  85 f1            sta attrib      akt. Attribut
12833  73c1
12834  73c1  20 d4 76         jsr defclr      SPRDEF Sprite Raster mit ac füllen
12835  73c4
12836  73c4
12837  73c4  ===> Ausgabe »sprite number ?« <===
12838  73c4
12839  73c4  a0 02      defbeg ldy #2          Spalte Ausgabe
12840  73c6  a2 17             ldx #23         Zeile Ausgabe
12841  73c8
12842  73c8  b9 6c 76   sprd30 lda stline-2,y  Text »sprite number?  «
12843  73cb  f0 06             beq sprd35      eom
12844  73cd
12845  73cd  20 db 68          jsr chrdsp      Zeichen in Bitspeicher bringen
12846  73d0  c8                iny
12847  73d1  d0 f5             bne sprd30      immer
12848  73d3
12849  73d3
12850  73d3  20 45 a8   sprd35 jsr mapio       I/O-Chips einblenden
12851  73d6
12852  73d6
12853  73d6  ===> Warten auf Eingabe Sprite Nummer <===
12854  73d6
12855  73d6  20 e4 ff   sprd40 jsr kgetin      Zeichen üb. Vector vom akt. Kanal
                                               nach ac (Sprungv.)
12856  73d9  f0 fb             beq sprd40      Warten auf Eingabe Sprite Nummer
12857  73db
12858  73db  c9 0d             cmp #_cr
12859  73dd  d0 08             bne sprd45
12860  73df
12861  73df  ===> vorzeitiger Abbruch SPRDEF <===
12862  73df
12863  73df  20 30 6b          jsr clrgrf      GRAPHIC RAM und Farb RAM löschen
12864  73e2
12865  73e2  a9 00             lda #%00000000  TEXT (GRAPH 0)
12866  73e4  85 d8             sta graphm      Flag Text/Graphik/Split-BS
12867  73e6
12868  73e6  60                rts
12869  73e7
12870  73e7
12871  73e7  38         sprd45 sec
12872  73e8  e9 31             sbc #'1'
12873  73ea  8d fc 12          sta spnumb      Sprite Nummer -1
12874  73ed
12875  73ed  c9 08             cmp #8
12876  73ef  b0 e5             bcs sprd40      Warten auf Eingabe Sprite Nummer
12877  73f1
12878  73f1  ===> DEFMOD Sprite bestimmen <===
12879  73f1
12880  73f1  aa                tax
```

```
12881  73f2  0a              asl a
12882  73f3  a8              tay
12883  73f4
12884  73f4  bd b3 6c        lda sbits,x       Tabelle Maskenbits Bit0-7
12885  73f7  8d 6d 11        sta bitmsk
12886  73fa
12887  73fa  2d 1c d0        and vicmcs        VIC 8564 Multi-Color-Modus Sprite7-0
12888  73fd
12889  73fd   76543210  MULTICOLOR Modus Sprite7-0 (1=ein)
12890  73fd
12891  73fd  f0 02           beq sprd50
12892  73ff
12893  73ff  a9 80           lda #%10000000    MULTICOLOR ein
12894  7401
12895  7401  8d fa 12   sprd50 sta defmod      Modus SPRDEF
12896  7404
12897  7404  ===> Sprite Koord. Kontrollausgabe <===
12898  7404
12899  7404  a9 08           lda #8
12900  7406  99 d6 11        sta postab,y      Abszisse Sprite
12901  7409
12902  7409  a9 4a           lda #74
12903  740b  99 d7 11        sta postab+1,y    Ordinate Sprite
12904  740e
12905  740e  ad 6d 11        lda bitmsk
12906  7411  0d e6 11        ora spmsb         MSB-Position Sprite7-0
12907  7414  8d e6 11        sta spmsb         MSB-Position Sprite7-0
12908  7417
12909  7417  ===> Sprite Ausgabe ein, Geschw. = 0 <===
12910  7417
12911  7417  ad 6d 11        lda bitmsk
12912  741a
12913  741a   76543210  1 = Sprite7-0 einschalten
12914  741a
12915  741a  8d 15 d0        sta vicmoe        VIC 8564 Sprite Enable Register
12916  741d
12917  741d  ae fc 12        ldx spnumb        Sprite Nummer -1
12918  7420  bc d9 6d        ldy sproff,x      Tabelle Offsets für Parameter
                                               Sprite0-7
12919  7423
12920  7423  a9 00           lda #0
12921  7425  99 7e 11        sta sspeed,y      Konstante Schritte / BASIC-IRQ
12922  7428
12923  7428  ===> Spritenummer ausgeben <===
12924  7428
12925  7428  8a              txa               Spritenummer -1
12926  7429
12927  7429  a0 11           ldy #17           Spalte
12928  742b  a2 17           ldx #23           Zeile
12929  742d
12930  742d  18              clc
12931  742e  69 31           adc #'1'          Sprite Nummer ASCII
12932  7430  20 db 68        jsr chrdsp        Zeichen in Bitspeicher bringen
12933  7433
12934  7433  ===> lstpnt auf Sprite Def. einstellen <===
12935  7433
12936  7433  20 45 a8        jsr mapio         I/O-Chips einblenden
12937  7436
```

```
12938  7436  ad fc 12          lda spnumb       Sprite Nummer -1
12939  7439  4a                lsr a            mal 64
12940  743a  6a                ror a
12941  743b  6a                ror a
12942  743c  85 4b             sta lstpnt       ADDRL Zeiger auf Sprite
12943  743e
12944  743e  a0 0e             ldy #>sprbas     ADDRH Sprite Basis Adresse
12945  7440  90 01             bcc sprd55
12946  7442  c8                iny              ADDRH Zeiger auf Sprite
12947  7443
12948  7443  84 4c       sprd55 sty lstpnt+1     ADDRH Zeiger auf Sprite
12949  7445
12950  7445  20 d1 75          jsr shwspr       Sprite auf Bildschirm darstellen
12951  7448
12952  7448  a0 3f             ldy #63
12953  744a  b1 4b       sprd60 lda (lstpnt),y  Sprite
12954  744c  99 b7 12          sta savram,y
12955  744f  88                dey
12956  7450  10 f8             bpl sprd60
12957  7452
12958  7452
12959  7452  ===> SPRDEF Cursor HOME <===
12960  7452
12961  7452  a2 00       dohome ldx #0
12962  7454  8e 5f 11          stx rowcnt       Zeile SPRDEF
12963  7457  8e 5e 11          stx colcnt       Spalte SPRDEF
12964  745a
12965  745a
12966  745a  ===> Eingabeschleife SPRDEF <===
12967  745a
12968  745a  20 4a 76    sprd65 jsr togcur      Cursor anzeigen
12969  745d
12970  745d
12971  745d  ===> Warten auf Eingabe <===
12972  745d
12973  745d  20 e4 ff    dfloop jsr kgetin      Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
12974  7460  f0 fb             beq dfloop       Warten auf Eingabe
12975  7462
12976  7462  48                pha              Eingabe Zeichen
12977  7463  20 4a 76          jsr togcur       Cursor anzeigen
12978  7466  68                pla              Eingabe Zeichen
12979  7467
12980  7467  ===> Test auf Vordergrundfarbe 1-16 <===
12981  7467
12982  7467  a2 10             ldx #16
12983  7469  dd b4 76    dloop1 cmp colnum-1,x  SPRDEF Tab. Tast. Codes Farben
12984  746c  d0 0d             bne dloop2       Zeiger auf nächsten Tast. Code
12985  746e
12986  746e  ===> Vordergrundfarbe eintragen <===
12987  746e
12988  746e  ca                dex
12989  746f  8a                txa              0-15
12990  7470
12991  7470  ae fc 12          ldx spnumb       Sprite Nummer -1
12992  7473  9d 27 d0          sta vicspc,x     VIC 8564 Farbe Sprite0
12993  7476
12994  7476  20 d1 75          jsr shwspr       Sprite auf Bildschirm darstellen
```

```
12995  7479  b0 df            bcs sprd65       immer
12996  747b
12997  747b
12998  747b
12998  747b  ===> Zeiger auf nächsten Tast. Code <===
12999  747b
13000  747b  ca         dloop2 dex
13001  747c  d0 eb             bne dloop1
13002  747e
13003  747e  ===> Test auf Tast. Code SPRDEF Befehl <===
13004  747e
13005  747e  a2 11              ldx #17
13006  7480  dd 7f 76   dloop3 cmp cmdstr,x     SPRDEF Tab. Tastatur Codes Befehle
13007  7483  f0 05              beq spdcmd       Aufruf Befehls Routine SPRDEF
13008  7485
13009  7485  ca                 dex
13010  7486  10 f8             bpl dloop3       Test nächster Tast. Code
13011  7488  30 d0             bmi sprd65       immer
13012  748a13013  748a
13014  748a  ===> Aufruf Befehls Routine SPRDEF <===
13015  748a
13016  748a  8a         spdcmd txa
13017  748b  a8                 tay              Zeiger auf SPRDEF Befehl
13018  748c
13019  748c  0a                 asl a
13020  748d  aa                 tax              Zeiger in Adreß Tabelle
13021  748e
13022  748e  bd 91 76           lda defjmp,x     SPRDEF Tab. Routinen Adressen
13023  7491  48                 pha
13024  7492
13025  7492  bd 92 76           lda defjmp+1,x   SPRDEF Tab. Routinen Adressen
13026  7495  48                 pha
13027  7496
13028  7496  60                 rts
13029  7497
13030  7497
13031  7497  ===> SPRDEF 1-4 (Farbquelle def.) <===
13032  7497
13033  7497  98         def040 tya
13034  7498  85 8f             sta vtemp2       Zeiger auf SPRDEF Befehl
13035  749a
13036  749a  20 10 76           jsr colbyt       Farbcode nach ac
13037  749d  48                 pha              Farbcode
13038  749e
13039  749e  ac 5e 11           ldy colcnt       Spalte SPRDEF
13040  74a1
13041  74a1  ae 5f 11           ldx rowcnt       Zeile
13042  74a4  20 c5 76           jsr getvm        Zeiger in GRAPHIC RAM auf Zeile xr
13043  74a7
13044  74a7  68                 pla              Farbcode
13045  74a8  20 3f 76           jsr setcol       ac nach GRAPHIC RAM
13046  74ab
13047  74ab  ===> Zeiger in Sprite Def. berechnen <===
13048  74ab
13049  74ab  ac 5e 11           ldy colcnt       Spalte SPRDEF
13050  74ae  98                 tya
13051  74af  29 07              and #%00000111
13052  74b1  aa                 tax              Spalte modulo 8
```

```
13053  74b2
13054  74b2  98              tya
13055  74b3  4a              lsr a
13056  74b4  4a              lsr a
13057  74b5  4a              lsr a           Spalte div. durch 8
13058  74b6
13059  74b6  18              clc
13060  74b7  6d 5f 11        adc rowcnt      Zeile SPRDEF
13061  74ba  6d 5f 11        adc rowcnt      Zeile SPRDEF
13062  74bd  6d 5f 11        adc rowcnt      Zeile SPRDEF
13063  74c0
13064  74c0  a8              tay             Zeiger in Sprite Def.
13065  74c1
13066  74c1  b1 4b           lda (lstpnt),y  Byte aus Sprite
13067  74c3
13068  74c3  2c fa 12        bit defmod      Modus SPRDEF
13069  74c6
13070  74c6   7....... MULTICOLOR SPRDEF (1=ein)
13071  74c6
13072  74c6  10 18           bpl def044
13073  74c8
13074  74c8  85 8e           sta vtemp1      Byte aus Sprite Def.
13075  74ca
13076  74ca  bd 1c 9d        lda rbits,x     Tab. Bits7-0
13077  74cd  1d 1d 9d        ora rbits+1,x   Tab. Bits7-0
13078  74d0  48              pha             Spalten-/Nachbar Spalten Bits
13079  74d1
13080  74d1  05 8e           ora vtemp1      in Sprite-Byte ODERn
13081  74d3  85 8e           sta vtemp1      Byte aus Sprite Def.
13082  74d5
13083  74d5  68              pla             Spalten-/Nachbar Spalten Bits
13084  74d6
13085  74d6  a6 8f           ldx vtemp2      Zeiger auf SPRDEF Befehl
                                             (=Farbquelle)
13086  74d8  3d 25 9f        and colval,x    Tab. Bitmuster Farbquelle Sprites
13087  74db  45 8e           eor vtemp1      Byte aus Sprite Def.
13088  74dd  4c ea 74        jmp def046
13089  74e0
13090  74e0
13091  74e0  1d 1c 9d  def044 ora rbits,x    Spalten Bit
13092  74e3  06 8f           asl vtemp2      Zeiger auf SPRDEF Befehl
                                             (=Farbquelle)
13093  74e5  d0 03           bne def046
13094  74e7
13095  74e7  ===> Graphik Modus 1 (Hintergrund Farbe) <===
13096  74e7
13097  74e7  5d 1c 9d        eor rbits,x     Tab. Bits7-0
13098  74ea
13099  74ea  91 4b     def046 sta (lstpnt),y Byte nach Sprite Def.
13100  74ec
13101  74ec  2c fa 12        bit defmod      Modus SPRDEF
13102  74ef
13103  74ef   .6......  0 = auto Cursor Vorschub
13104  74ef
13105  74ef  50 4e           bvc def110      SPRDEF CRIGHT (Cursor rechts)
13106  74f1
13107  74f1  4c 5a 74        jmp sprd65      Eingabeschleife SPRDEF
13108  74f4
```

```
13109  74f4
13110  74f4
13110  74f4  ===> SPRDEF STOPP Taste (altes Sprite zurück) <===
13111  74f4
13112  74f4  a0 3f      def050 ldy #63
13113  74f6  b9 b7 12   def051 lda savram,y
13114  74f9  91 4b            sta (lstpnt),y   Sprite Def.
13115  74fb  88               dey
13116  74fc  10 f8            bpl def051
13117  74fe
13118  74fe
13119  74fe  ===> SPRDEF SHIFT CR (Ende SPRDEF) <===
13120  74fe
13121  74fe  a9 00      def060 lda #%00000000
13122  7500
13123  7500   76543210  1 = Sprite7-0 einschalten
13124  7500
13125  7500  8d 15 d0          sta vicmoe       VIC 8564 Sprite Enable Register
13126  7503
13127  7503  4c c4 73          jmp defbeg       Ausgabe »sprite number ?«
13128  7506
13129  7506
13130  7506  ===> SPRDEF M (Zusatzfarbe ein/aus) <===
13131  7506
13132  7506  ad fa 12   def070 lda defmod       Modus SPRDEF
13133  7509  49 80             eor #%10000000   MULTICOLOR Bit
13134  750b  8d fa 12          sta defmod       Modus SPRDEF
13135  750e
13136  750e  20 d1 75          jsr shwspr       Sprite auf Bildschirm darstellen
13137  7511
13138  7511  ad 5e 11          lda colcnt       Spalte SPRDEF
13139  7514  29 fe             and #%11111110   gerade
13140  7516  8d 5e 11          sta colcnt       Spalte SPRDEF
13141  7519
13142  7519  a0 1c             ldy #28          MULTICOLOR Modus Reg.
13143  751b
13144  751b   76543210  MULTICOLOR Sprite7-0 (1=ein)
13145  751b
13146  751b  2c          .byte  _skip2
13147  751c
13148  751c
13149  751c  ===> SPRDEF Expand y-Richtung <===
13150  751c
13151  751c  a0 17      def080 ldy #23          Expand Sprite Vert. Reg.
13152  751e
13153  751e   76543210  Sprite7-0 vert. verdoppeln (1=ein)
13154  751e
13155  751e  2c                .byte _skip2
13156  751f
13157  751f
13158  751f  ===> SPRDEF Expand x-Richtung <===
13159  751f
13160  751f  a0 1d      def090 ldy #29          Expand Sprite Hor. Reg.
13161  7521
13162  7521   76543210  Sprite7-0 hor. verdoppeln (1=ein)
13163  7521
```

```
13164  7521  ===> Sprite Bit flippen <===
13165  7521
13166  7521  b9 00 d0          lda vic,y       VIC 8564 Basisadresse
13167  7524  ae fc 12          ldx spnumb      Sprite Nummer -1
13168  7527  5d b3 6c          eor sbits,x     Tabelle Maskenbits Bit0-7
13169  752a  99 00 d0          sta vic,y       VIC 8564 Basisadresse
13170  752d
13171  752d  4c 5a 74          jmp sprd65      Eingabeschleife SPRDEF
13172  7530
13173  7530
13174  7530  ===> SPRDEF CLR (Sprite löschen) <===
13175  7530
13176  7530  a0 3f      def100 ldy #63
13177  7532  a9 00             lda #0
13178  7534  91 4b      def102 sta (lstpnt),y  Sprite Def.
13179  7536  88               dey
13180  7537  10 fb             bpl def102
13181  7539
13182  7539  20 d4 76          jsr defclr      SPRDEF Sprite Raster mit ac füllen
13183  753c  4c 52 74          jmp dohome      SPRDEF Cursor HOME
13184  753f
13185  753f
13186  753f  ===> SPRDEF CRIGHT (Cursor rechts) <===
13187  753f
13188  753f  a9 01      def110 lda #1
13189  7541  2c              .byte  _skip2
13190  7542
13191  7542
13192  7542  ===> SPRDEF CLEFT (Cursor links) <===
13193  7542
13194  7542  a9 ff      def120 lda #$ff         - 1
13195  7544
13196  7544  2c fa 12          bit defmod      Multicolor-Modus?
13197  7547
13198  7547   7.......  1 = MULTICOLOR Sprite
13199  7547
13200  7547  10 01             bpl def121
13201  7549
13202  7549  0a                asl a
13203  754a
13204  754a  18        def121 clc
13205  754b  6d 5e 11          adc colcnt      Spalte SPRDEF
13206  754e  30 09             bmi def122      vorige Zeile rechte Rand Spalte
13207  7550
13208  7550  c9 18             cmp #23+1       rechte Rand Spalte
13209  7552  b0 2d             bcs def160      SPRDEF CR (Zeilenvorschub)
13210  7554
13211  7554  8d 5e 11          sta colcnt      Spalte SPRDEF
13212  7557  90 25             bcc def155      immer
13213  7559
13214  7559
13215  7559  ===> vorige Zeile rechte Rand Spalte <===
13216  7559
13217  7559  a2 17      def122 ldx #23         rechte Rand Spalte
13218  755b
13219  755b  2c fa 12          bit defmod      Modus SPRDEF
13220  755e
13221  755e   7.......  1 = MULTICOLOR Sprite
```

```
13222  755e
13223  755e  10 01            bpl def123
13224  7560
13225  7560  ca               dex             Spalte SPRDEF
13226  7561
13227  7561  8e 5e 11  def123 stx colcnt      Spalte SPRDEF
13228  7564
13229  7564
13230  7564  ===> SPRDEF CRSUP (Cursor n. oben) <===
13231  7564
13232  7564  a9 ff     def140 lda #$ff
13233  7566  2c               .byte _skip2
13234  7567
13235  7567
13236  7567  ===> SPRDEF CRSDOWN (Cursor n. unten) <===
13237  7567
13238  7567  a9 01     def130 lda #1
13239  7569
13240  7569  18               clc
13241  756a  6d 5f 11         adc rowcnt      Zeile SPRDEF
13242  756d
13243  756d  c9 15            cmp #20+1       Untergrenze
13244  756f  b0 0d            bcs def155
13245  7571
13246  7571  8d 5f 11         sta rowcnt      Zeile SPRDEF
13247  7574  90 08            bcc def155      immer
13248  7576
13249  7576
13250  7576  ===> SPRDEF A (auto Cursor Vorschub) <===
13251  7576
13252  7576  ad fa 12  def150 lda defmod      Modus SPRDEF
13253  7579  49 40            eor #%01000000  auto Cursor Vorschub Bit
13254  757b  8d fa 12         sta defmod      Modus SPRDEF
13255  757e
13256  757e  4c 5a 74  def155 jmp sprd65      Eingabeschleife SPRDEF
13257  7581
13258  7581
13259  7581  ===> SPRDEF CR (Zeilenvorschub) <===
13260  7581
13261  7581  a9 00     def160 lda #0
13262  7583  8d 5e 11         sta colcnt      Spalte SPRDEF
13263  7586  f0 df            beq def130      immer
13264  7588
13265  7588
13266  7588  ===> SPRDEF C (Sprite kopieren) <===
13267  7588
13268  7588  a0 02     defcpy ldy #2
13269  758a  a2 18              ldx #24
13270  758c  b9 61 76  defcp1 lda coplin-2,y  Text »copy from?«
13271  758f  f0 06            beq defcp2      eom
13272  7591
13273  7591  20 db 68         jsr chrdsp      Zeichen in Bitspeicher bringen
13274  7594  c8               iny
13275  7595  d0 f5            bne defcp1      immer
13276  7597
13277  7597
```

```
13278  7597  ===> Eingabeschleife SPRDEF COPY <===
13279  7597
13280  7597  20 45 a8   defcp2 jsr mapio        I/O-Chips einblenden
13281  759a
13282  759a
13283  759a
13283  759a  ===> Warten auf Eingabe <===
13284  759a
13285  759a  20 e4 ff   defcp3 jsr kgetin       Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
13286  759d  f0 fb             beq defcp3       Warten auf Eingabe
13287  759f
13288  759f  c9 0d             cmp #_cr
13289  75a1  f0 1f             beq defcp6
13290  75a3
13291  75a3  38                sec
13292  75a4  e9 31             sbc #'1'         Sprite7-0
13293  75a6
13294  75a6  c9 08             cmp #8
13295  75a8  b0 f0             bcs defcp3       Warten auf Eingabe
13296  75aa
13297  75aa  4a                lsr a            mal 64
13298  75ab  6a                ror a
13299  75ac  6a                ror a
13300  75ad  85 8e             sta vtemp1       ADDRL Sprite Def.
13301  75af
13302  75af  a0 0e             ldy #>sprbas     ADDRH Sprite Basis Adresse
13303  75b1  90 01             bcc defcp4
13304  75b3  c8                iny              ADDRH Sprite Basis Adresse
13305  75b4  84 8f      defcp4 sty vtemp2       ADDRH Sprite Def.
13306  75b6
13307  75b6  a0 3f             ldy #63
13308  75b8  b1 8e      defcp5 lda (vtemp1),y   Sprite Def.
13309  75ba  91 4b             sta (lstpnt),y   Sprite Kopie
13310  75bc  88                dey
13311  75bd  10 f9             bpl defcp5
13312  75bf
13313  75bf  20 d1 75          jsr shwspr       Sprite auf Bildschirm darstellen
13314  75c2
13315  75c2  a9 00      defcp6 lda #0
13316  75c4  a8                tay
13317  75c5  99 00 3e   defcp7 sta gbasic-512,y
13318  75c8  88                dey
13319  75c9  d0 fa             bne defcp7
13320  75cb
13321  75cb  4c 5a 74          jmp sprd65       Eingabeschleife SPRDEF
13322  75ce
13323  75ce
13324  75ce  ff ff ff          .byte $ff, $ff, $ff
13325  75d1
13326  75d1
13327  75d1  ===> Sprite auf Bildschirm darstellen <===
13328  75d1
13329  75d1  a2 00      shwspr ldx #0
13330  75d3  8e 60 11          stx strcnt       Zeiger in Sprite Def.
13331  75d6
13332  75d6  8e fb 12   shwsp1 stx lincnt       Zeilen Zähler
13333  75d9  20 c5 76          jsr getvm        Zeiger in GRAPHIC RAM auf Zeile xr
```

```
13334  75dc
13335  75dc  a0 00           ldy #0
13336  75de  a2 08     shwsp2 ldx #8          Bit Zähler
13337  75e0  8c 6e 11        sty numcnt       Zeichenzähler
13338  75e3
13339  75e3  ac 60 11        ldy strcnt       Zeiger in Sprite Def.
13340  75e6  b1 4b           lda (lstpnt),y   Byte aus Sprite nach AC
13341  75e8  ee 60 11        inc strcnt       Zeiger in Sprite Def.
13342  75eb
13343  75eb  ac 6e 11        ldy numcnt       Zeichenzähler
13344  75ee  0a       shwsp3 asl a
13345  75ef
13346  75ef  2c fa 12        bit defmod       Modus SPRDEF
13347  75f2
13348  75f2   7.......  1 = MULTICOLOR Sprite
13349  75f2
13350  75f2  10 02           bpl shwsp4
13351  75f4
13352  75f4  2a              rol a
13353  75f5  ca              dex              Bit Zähler
13354  75f6
13355  75f6  48       shwsp4 pha
13356  75f7  2a              rol a
13357  75f8  20 10 76        jsr colbyt       Farbcode nach ac
13358  75fb  20 3f 76        jsr setcol       ac nach GRAPHIC RAM
13359  75fe
13360  75fe  c8              iny
13361  75ff  68              pla
13362  7600
13363  7600  ca              dex              Bit Zähler
13364  7601  d0 eb           bne shwsp3
13365  7603
13366  7603  c0 18           cpy #24
13367  7605  90 d7           bcc shwsp2
13368  7607
13369  7607  ae fb 12        ldx lincnt       Zeilen Zähler
13370  760a  e8              inx              erhöhen
13371  760b
13372  760b  e0 15           cpx #$15         alle Zeilen durch?
13373  760d  90 c7           bcc shwsp1       nein
13374  760f
13375  760f  60              rts
13376  7610
13377  7610
13378  7610  ===> Farbcode nach ac <===
13379  7610
13380  7610  29 03     colbyt and #%00000011
13381  7612  4a              lsr a
13382  7613  6a              ror a
13383  7614  f0 0f           beq colbt1       Hintergrund Farbe
13384  7616
13385  7616  2c fa 12        bit defmod       Modus SPRDEF
13386  7619
13387  7619   7.......  1 = MULTICOLOR Sprite
13388  7619
13389  7619  10 0f           bpl colbt2
13390  761b
```

```
13391  761b  ===> MULTICOLOR 1 <===
13392  761b
13393  761b  ad 25 d0          lda vicmc0       VIC 8564 Sprite Multicolor #0
13394  761e  90 14             bcc colbt3
13395  7620
13396  7620
13396  7620  ===> MULTICOLOR 2 <===
13397  7620
13398  7620  ad 26 d0          lda vicmc1       VIC 8564 Sprite Multicolor #1
13399  7623  b0 0f             bcs colbt3       immer
13400  7625
13401  7625
13402  7625  ===> Hintergrund Farbe <===
13403  7625
13404  7625  ad 21 d0   colbt1 lda vicbkg       VIC 8564 Hintergrundfarbe
13405  7628  90 0a             bcc colbt3
13406  762a
13407  762a  86 8e       colbt2 stx vtemp1      xr
13408  762c
13409  762c  ae fc 12           ldx spnumb       Sprite Nummer -1
13410  762f  bd 27 d0           lda vicspc,x     VIC 8564 Farbe Sprite0
13411  7632
13412  7632  a6 8e              ldx vtemp1       xr
13413  7634
13414  7634  29 0f       colbt3 and #%00001111
13415  7636  85 8e              sta vtemp1
13416  7638
13417  7638  0a                 asl a
13418  7639  0a                 asl a
13419  763a  0a                 asl a
13420  763b  0a                 asl a
13421  763c  05 8e              ora vtemp1
13422  763e  60                 rts
13423  763f
13424  763f
13425  763f  ===> ac nach GRAPHIC RAM <===
13426  763f
13427  763f  91 8c       setcol sta (grapnt),y  ADDRL Zeiger in Graphik-RAM
13428  7641
13429  7641  2c fa 12           bit defmod       Multicolor-Modus?
13430  7644
13431  7644   7.......  1 = MULTICOLOR Sprite
13432  7644
13433  7644  10 03              bpl setco1
13434  7646
13435  7646  c8                 iny
13436  7647  91 8c              sta (grapnt),y   ADDRL Zeiger in Graphik-RAM
13437  7649
13438  7649  60          setco1 rts
13439  764a
13440  764a
13441  764a  ===> Cursor anzeigen <===
13442  764a
13443  764a  ae 5f 11   togcur ldx rowcnt       Zeile SPRDEF
13444  764d  20 c5 76          jsr getvm        Zeiger in GRAPHIC RAM auf Zeile xr
13445  7650
13446  7650  ac 5e 11          ldy colcnt       Spalte SPRDEF
13447  7653
```

```
13448  7653  2c fa 12          bit defmod       Modus SPRDEF
13449  7656
13450  7656   7.......  1 = MULTICOLOR Sprite
13451  7656
13452  7656  10 03             bpl makcr1
13453  7658
13454  7658  20 5b 76          jsr makcr1
13455  765b
13456  765b  b1 8c       makcr1 lda (grapnt),y   ADDRL Zeiger in Graphik-RAM
13457  765d  49 80              eor #%10000000
13458  765f  91 8c              sta (grapnt),y   ADDRL Zeiger in Graphik-RAM
13459  7661
13460  7661  c8                 iny
13461  7662  60                 rts
13462  7663
13463  7663
13464  7663  ===> Text »copy from?« <===
13465  7663
13466  7663  43 4f 50    coplin .byte 'copy from?', _eom
13466  7666  59 20 46
13466  7669  52 4f 4d
13466  766c  3f 00
13467  766e13468  766e
13469  766e  ===> Text »sprite number?  « <===
13470  766e
13471  766e  53 50 52    stline .byte 'sprite number?  ', _eom
13471  7671  49 54 45
13471  7674  20 4e 55
13471  7677  4d 42 45
13471  767a  52 3f 20
13471  767d  20 00
13472  767f
13473  767f
13474  767f  ===> SPRDEF Tab. Tastatur Codes Befehle <===
13475  767f
13476  767f  31 32 33    cmdstr .byte '1234', _stpky, _sftcr, 'xym'
13476  7682  34 03 8d
13476  7685  58 59 4d
13477  7688  9d 1d 91           .byte _cleft, _crite, _crsup, _crsdn, _clr, _home
13477  768b  11 93 13
13478  768e  41 0d 43           .byte 'a', _cr, 'c'
13479  7691
13480  7691
13481  7691  ===> SPRDEF Tab. Routinen Adressen <===
13482  7691
13483  7691  74 96       defjmp .dbyte def040-1  SPRDEF 1-4 (Farbquelle def.)
13484  7693  74 96              .dbyte def040-1  SPRDEF 1-4 (Farbquelle def.)
13485  7695  74 96              .dbyte def040-1  SPRDEF 1-4 (Farbquelle def.)
13486  7697  74 96              .dbyte def040-1  SPRDEF 1-4 (Farbquelle def.)
13487  7699  74 f3              .dbyte def050-1  SPRDEF STOPP Taste (altes Sprite
                                                zurück)
13488  769b  74 fd              .dbyte def060-1  SPRDEF SHIFT CR (Ende SPRDEF)
13489  769d  75 1e              .dbyte def090-1  SPRDEF Expand x-Richtung
13490  769f  75 1b              .dbyte def080-1  SPRDEF Expand y-Richtung
13491  76a1  75 05              .dbyte def070-1  SPRDEF M (Zusatzfarbe ein/aus)
13492  76a3  75 41              .dbyte def120-1  SPRDEF CLEFT (Cursor links)
13493  76a5  75 3e              .dbyte def110-1  SPRDEF CRIGHT (Cursor rechts)
13494  76a7  75 63              .dbyte def140-1  SPRDEF CRSUP (Cursor n. oben)
```

```
13495  76a9  75 66           .dbyte def130-1  SPRDEF CRSDOWN (Cursor n. unten)
13496  76ab  75 2f           .dbyte def100-1  SPRDEF CLR (Sprite löschen)
13497  76ad  74 51           .dbyte dohome-1  SPRDEF Cursor HOME
13498  76af  75 75           .dbyte def150-1  SPRDEF A (auto Cursor Vorschub)
13499  76b1  75 80           .dbyte def160-1  SPRDEF CR (Zeilenvorschub)
13500  76b3  75 87           .dbyte defcpy-1  SPRDEF C (Sprite kopieren)
13501  76b5
13502  76b5
13503  76b5  ===> SPRDEF Tab. Tast. Codes Farben <===
13504  76b5
13505  76b5  90 05 1c   colnum .byte $90, $05, $1c, $9f
13505  76b8  9f
13506  76b9  9c 1e 1f         .byte $9c, $1e, $1f, $9e
13506  76bc  9e
13507  76bd  81 95 96         .byte $81, $95, $96, $97
13507  76c0  97
13508  76c1  98 99 9a         .byte $98, $99, $9a, $9b
13508  76c4  9b
13509  76c5
13510  76c5
13511  76c5  ===> Zeiger in GRAPHIC RAM auf Zeile xr <===
13512  76c5
13513  76c5  bd 33 c0   getvm  lda ldtb2,x      Tab. ADDRL BS Zeilen
13514  76c8  85 8c            sta grapnt       ADDRL Zeiger in Graphik-RAM
13515  76ca
13516  76ca  bd 4c c0         lda ldtb1,x      Tab. ADDRH BS Zeilen
13517  76cd  29 03            and #%00000011
13518  76cf  09 1c            ora #%00011100
13519  76d1  85 8d            sta grapnt+1     ADDRH Zeiger in Graphik-RAM
13520  76d3
13521  76d3  60               rts
13522  76d4
13523  76d4
13524  76d4  ===> SPRDEF Sprite Raster mit ac füllen <===
13525  76d4
13526  76d4  ad 21 d0   defclr lda vicbkg       VIC 8564 Hintergrundfarbe
13527  76d7  20 34 76         jsr colbt3       nach ob. Nibble
13528  76da
13529  76da  a2 14            ldx #20
13530  76dc  48         defcl1 pha
13531  76dd  20 c5 76         jsr getvm        Zeiger in GRAPHIC RAM auf Zeile xr
13532  76e0  68               pla
13533  76e1
13534  76e1  a0 17            ldy #23
13535  76e3  91 8c      defcl2 sta (grapnt),y   ADDRL Zeiger in Graphik-RAM
13536  76e5  88               dey
13537  76e6  10 fb            bpl defcl2
13538  76e8
13539  76e8  ca               dex
13540  76e9  10 f1            bpl defcl1
13541  76eb
13542  76eb  60               rts
13543  76ec
13544  76ec
13545  76ec  ===> BASIC-Statement SPRSAV ($fe $16) <===
13546  76ec
13547  76ec  20 7c 77   sprsav jsr savinp       Sprite Nummer / -String auswerten
13548  76ef  b0 2f            bcs sprs20       Sprite String nach Sprite Def.
```

```
13549  76f1
13550  76f1  ===> Sprite Def. nach Sprite String <===
13551  76f1
13552  76f1  85 4b           sta lstpnt       ADDRL Zeiger Sprite Def.
13553  76f3  84 4c           sty lstpnt+1     ADDRH Zeiger Sprite Def.
13554  76f5
13555  76f5
13555  76f5  ===> Sprite Def. kopieren <===
13556  76f5
13557  76f5  a0 3e           ldy #62
13558  76f7  b1 4b     sprs10 lda (lstpnt),y   Sprite Def.
13559  76f9  99 b7 12         sta savram,y
13560  76fc  88               dey
13561  76fd  10 f8            bpl sprs10
13562  76ff
13563  76ff  c8               iny              0
13564  7700  8c f7 12         sty savram+64    High-Byte Breite
13565  7703  8c f9 12         sty savram+66    High-Byte Höhe
13566  7706
13567  7706  a9 17            lda #23
13568  7708  8d f6 12         sta savram+63    Low-Byte Breite
13569  770b
13570  770b  a9 14            lda #20
13571  770d  8d f8 12         sta savram+65    Low-Byte Höhe
13572  7710
13573  7710  a2 b7            ldx #<savram
13574  7712  a0 12            ldy #>savram
13575  7714
13576  7714  86 70            stx strng1
13577  7716  84 71            sty strng1+1
13578  7718
13579  7718  a9 43            lda #67          Stringlänge
13580  771a  20 cc 86         jsr strlt1       ac Bytes strng1 ind. nach Arb.String
13581  771d  20 99 77         jsr dscfre       String Descr. nach xr,ac,yr
13582  7720
13583  7720  8e db 03  sprs20 stx savsiz       String Länge
13584  7723  8d dc 03         sta savsiz+1     ADDRL String
13585  7726  8c dd 03         sty savsiz+2     ADDRH String
13586  7729
13587  7729  ===> zweiten Parameter auswerten <===
13588  7729
13589  7729  20 5c 79         jsr chkcom       Test auf folg. »Komma« sonst Fehler
13590  772c
13591  772c  a5 3d            lda txtptr       ADDRL chrget-Zeiger
13592  772e  8d e0 03         sta sptmp1
13593  7731
13594  7731  a5 3e            lda txtptr+1     ADDRH chrget-Zeiger
13595  7733  8d e1 03         sta sptmp2
13596  7736
13597  7736  20 7c 77         jsr savinp       Sprite Nummer / -String auswerten
13598  7739  b0 25            bcs sprs50       String
13599  773b
13600  773b  85 8c            sta grapnt       ADDRL Zeiger Sprite Def.
13601  773d  84 8d            sty grapnt+1     ADDRH Zeiger Sprite Def.
13602  773f
13603  773f  ad dc 03         lda savsiz+1     ADDRL String
13604  7742  85 4b            sta lstpnt       ADDRL String Zeiger
13605  7744
```

```
13606  7744  ad dd 03          lda savsiz+2     ADDRH String
13607  7747  85 4c             sta lstpnt+1     ADDRH String Zeiger
13608  7749
13609  7749  a0 00             ldy #0
13610  774b  cc db 03   sprs30 cpy savsiz       Stringende erreicht?
13611  774e  f0 0f             beq sprs40       ja
13612  7750
13613  7750  a9 4b             lda #<lstpnt     String
13614  7752  20 ab 03          jsr iram1        LDA (ac),y aus VAR-RAM
13615  7755
13616  7755  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
13617  7758
13618  7758  91 8c             sta (grapnt),y   Sprite Def.
13619  775a  c8                iny
13620  775b
13621  775b  c0 3f             cpy #63
13622  775d  d0 ec             bne sprs30
13623  775f
13624  775f  60          sprs40 rts
13625  7760
13626  7760
13627  7760  ad e0 03    sprs50 lda sptmp1
13628  7763  85 3d             sta txtptr       ADDRL chrget-Zeiger
13629  7765
13630  7765  ad e1 03          lda sptmp2
13631  7768  85 3e             sta txtptr+1     ADDRH chrget-Zeiger
13632  776a
13633  776a  20 af 7a          jsr ptrget       folg. Variable suchen/anlegen
13634  776d  85 4b             sta lstpnt       ADDRL Zeiger auf Variable
13635  776f  84 4c             sty lstpnt+1     ADDRH Zeiger auf Variable
13636  7771
13637  7771  a9 db             lda #<savsiz     ADDRL String Descriptor
13638  7773  85 66             sta indice
13639  7775
13640  7775  a9 03             lda #>savsiz     ADDRH String Descriptor
13641  7777  85 67             sta indice+1
13642  7779
13643  7779  4c 05 54          jmp inpcom       String zuweisen
13644  777c
13645  777c
13646  777c  ===> Sprite Nummer / -String auswerten <===
13647  777c
13648  777c  20 ef 77   savinp jsr frmevl       folgenden Ausdruck auswerten
13649  777f
13650  777f  24 0f             bit valtyp       Flag Variablentyp
13651  7781  30 16             bmi dscfre       String
13652  7783
13653  7783  ===> Sprite Nummer einlesen <===
13654  7783
13655  7783  20 f7 87          jsr conint       8 Bit INT CONST aus FAC nach xr
13656  7786  ca                dex              Sprite Nummer -1
13657  7787
13658  7787  e0 08             cpx #8
13659  7789  b0 0b             bcs fcerr3       Behandlung »?illegal quantity error«
13660  778b
13661  778b  8a                txa              Sprite Nummer -1
13662  778c  4a                lsr a            mal 64
13663  778d  6a                ror a
```

```
13664  778e  6a              ror a
13665  778f
13666  778f  a0 0e           ldy #>sprbas     ADDRH Sprite Basis Adresse
13667  7791  90 01           bcc savin1
13668  7793  c8              iny              ADDRH Zeiger auf Sprite
13669  7794
13670  7794  18       savin1 clc              Flag numer. Var.
13671  7795  60              rts
13672  7796
13673  7796
13674  7796  ===> Behandlung »?illegal quantity error« <===
13675  7796
13676  7796  4c 28 7d  fcerr3 jmp fcerr       Behandlung »?illegal quantity error«
13677  7799
13678  7799
13679  7799  ===> String Descr. nach xr,ac,yr <===
13680  7799
13681  7799  a5 66     dscfre lda indice
13682  779b  a4 67            ldy indice+1
13683  779d  20 e0 87         jsr fretms       Descriptor vom Stapel entfernen
13684  77a0
13685  77a0  a0 00            ldy #0
13686  77a2  20 e7 42         jsr indfmo       LDA (indice),y aus VAR RAM
13687  77a5  aa               tax              String Länge
13688  77a6
13689  77a6  c8              iny
13690  77a7  20 e7 42         jsr indfmo       LDA (indice),y aus VAR RAM
13691  77aa  48               pha              ADDRL String
13692  77ab
13693  77ab  c8              iny
13694  77ac  20 e7 42         jsr indfmo       LDA (facmo),y aus VAR-RAM
13695  77af  a8               tay              ADDRH String
13696  77b0
13697  77b0  68               pla              addrl Stringadresse
13698  77b1
13699  77b1  38               sec              Flag für String
13700  77b2  60               rts
13701  77b3
13702  77b3
13703  77b3  ===> BASIC-Statement FAST ($fe $25) <===
13704  77b3
13705  77b3  20 45 a8   fast   jsr mapio       I/O-Chips einblenden
13706  77b6
13707  77b6  ad 11 d0           lda viccr1       VIC 8564 Control-Register 1
13708  77b9
13709  77b9   7.......  MSB (=Bit8) Raster Wert Reg. 18
13710  77b9   ...4....  Blank Screen to Color Border (0=blank)
13711  77b9
13712  77b9  29 6f            and #%01101111   BS blank
13713  77bb  8d 11 d0         sta viccr1       VIC 8564 Control-Register 1
13714  77be
13715  77be  a9 01            lda #%00000001
13716  77c0
13717  77c0   7654321.  unbenutzt
13718  77c0   .......0  Flag CPU Takt (0=1MHz, 1=2MHz)
13719  77c0
13720  77c0  8d 30 d0         sta vicspd       VIC 8564 Speed-Register
13721  77c3  60               rts
```

```
13722  77c4
13723  77c4
13724  77c4  ===> BASIC-Statement SLOW ($fe $26) <===
13725  77c4
13726  77c4  20 45 a8   slow   jsr mapio        I/O-Chips einblenden
13727  77c7
13728  77c7  a9 00             lda #%00000000
13729  77c9
13730  77c9   7654321.  unbenutzt
13731  77c9   .......0  Flag CPU Takt (0=1MHz, 1=2MHz)
13732  77c9
13733  77c9  8d 30 d0          sta vicspd       VIC 8564 Speed-Register
13734  77cc
13735  77cc  ad 11 d0          lda viccr1       VIC 8564 Control-Register 1
13736  77cf
13737  77cf   7.......  MSB (=Bit8) Raster Wert Reg. 18
13738  77cf   ...4....  Blank Screen to Color Border (0=blank)
13739  77cf
13740  77cf  29 7f             and #%01111111
13741  77d1  09 10             ora #%00010000
13742  77d3  8d 11 d0          sta viccr1       VIC 8564 Control-Register 1
13743  77d6
13744  77d6  60                rts
13745  77d7
13746  77d7
13747  77d7                    .end
13748  77d7                    .lib basic.eval
13749  77d7  put"@basic.eval"
```

```
13751  77d7
13752  77d7
13753  77d7  ===> folg. num. Variable nach FAC <===
13754  77d7
13755  77d7  20 ef 77    frmnum jsr frmevl        folgenden Ausdruck auswerten
13756  77da
13757  77da
13758  77da  ===> Test num. VAR, sonst type mism. <===
13759  77da
13760  77da  18          chknum clc
13761  77db  90 01              bcc chkval        Test VAR Typ (CARRY=1:String,
                                                 =0:num)
13762  77dd
13763  77dd
13764  77dd  ===> Test String VAR, sonst type mism. <===
13765  77dd
13766  77dd  38          chkstr sec
13767  77de
13768  77de
13769  77de  ===> Test VAR Typ (CARRY=1:String, =0:num) <===
13770  77de
13771  77de  24 0f       chkval bit valtyp        Flag Variablentyp
13772  77e0  30 03              bmi docstr        Variable ist String
13773  77e2  b0 03              bcs chkerr        Behandlung »?type mismatch error«
13774  77e4
13775  77e4  60          chkok  rts
13776  77e5
13777  77e5
13778  77e5  ===> Variable ist String <===
13779  77e5
13780  77e5  b0 fd       docstr bcs chkok
13781  77e7
13782  77e7
13783  77e7  ===> Behandlung »?type mismatch error« <===
13784  77e7
13785  77e7  a2 16       chkerr ldx #_ertm
13786  77e9  2c                 .byte _skip2
13787  77ea
13788  77ea
13789  77ea  ===> Behandlung »?formula too complex error« <===
13790  77ea
13791  77ea  a2 19       sterr  ldx #_erst
13792  77ec  4c 3c 4d           jmp error         ind. Behandlung Fehlermeldung
13793  77ef
13794  77ef
13795  77ef  ===> folgenden Ausdruck auswerten <===
13796  77ef
13797  77ef  a6 3d       frmevl ldx txtptr        ADDRL chrget-Zeiger
13798  77f1  d0 02              bne frmev1        zurücksetzen
13799  77f3  c6 3e              dec txtptr+1      ADDRH chrget-Zeiger
13800  77f5  c6 3d       frmev1 dec txtptr        ADDRL chrget-Zeiger
13801  77f7
13802  77f7  a2 00              ldx #$00          Prioritäts Flag Abschluß
13803  77f9
13804  77f9   Funktionen werden nach Priorität bewertet (PRIO Flag).
13805  77f9   Funktionen mit höherer Priorität werden zuerst ausgeführt,
13806  77f9   solche mit niedrigerer legen die Adresse ihrer System-
13807  77f9   routine, den FAC und ihr Prioritätsflag auf den Stapel
```

```
13808  77f9   (dopre1). Wird das PRIO Flag = 0 vom Stapel geholt, ist
13809  77f9   der Ausdruck ausgewertet: numerischer Wert bzw. String-
13810  77f9   adresse findet man in ARG
13811  77f9
13812  77f9  24              .byte _skip1
13813  77fa
13814  77fa
13815  77fa  ===> Auswerte Schleife <===
13816  77fa
13817  77fa  48        lpoper pha              Operator Maske
13818  77fb
13819  77fb  8a               txa
13820  77fc  48               pha              Prioritäts Flag
13821  77fd
13822  77fd  ba               tsx
13823  77fe  e0 63            cpx #99
13824  7800  90 e8            bcc sterr        Behandlung »?formula too complex
                                                error«
13825  7802
13826  7802  20 d7 78         jsr eval         ind. folg. Ausdruck auswerten
13827  7805
13828  7805  a9 00            lda #%00000000   rücksetzen
13829  7807
13830  7807   76543...  nicht benutzt
13831  7807   .....2..  1 = KLEINER (4)
13832  7807   ......1.  1 = GLEICH  (2)
13833  7807   .......0  1 = GRÖSSER (1)
13834  7807
13835  7807  85 4f            sta opmask       vom akt. Operator erzeugte Bit-Maske
13836  7809
13837  7809  20 86 03   tstop  jsr chrgot       letztes Zeichen erneut nach ac
13838  780c
13839  780c
13840  780c  ===> Test / Bearb. GRÖSSER GLEICH KLEINER <===
13841  780c
13842  780c  38        loprel sec
13843  780d  e9 b1            sbc #_great      GRÖSSER Token
13844  780f  90 17            bcc endrel
13845  7811
13846  7811  c9 03            cmp #_onefn-_great
13847  7813  b0 13            bcs endrel       nicht GRÖSSER GLEICH KLEINER
13848  7815
13849  7815  ===> Behandlung GRÖSSER GLEICH KLEINER <===
13850  7815
13851  7815   AC: 0 = GRÖSSER
13852  7815       1 = GLEICH
13853  7815       2 = KLEINER
13854  7815
13855  7815  c9 01            cmp #1           CLC bei GRÖSSER
13856  7817  2a               rol a            CARRY nach Bit0
13857  7818  49 01            eor #%00000001   SET Bit0 GRÖSSER, sonst RESET
13858  781a  45 4f            eor opmask       RESET Bit beim 2. Mal gl. Operator
13859  781c
13860  781c   76543...  nicht benutzt
13861  781c   .....2..  1 = KLEINER (4)
13862  781c   ......1.  1 = GLEICH  (2)
13863  781c   .......0  1 = GRÖSSER (1)
13864  781c
```

```
13865  781c
13866  781c
13866  781c  ===> Test gl. Operator zweimal <===
13867  781c
13868  781c  c5 4f            cmp opmask      vom akt. Operator erzeugte Bit-Maske
13869  781e  90 61            bcc snerr5      gl. Operator zweimal
13870  7820
13871  7820  ===> Operator Maske abspeichern <===
13872  7820
13873  7820  85 4f            sta opmask      vom akt. Operator erzeugte Bit-Maske
13874  7822  20 80 03         jsr chrget      nächstes Zeichen nach ac
13875  7825  4c 0c 78         jmp loprel      Test / Bearb. GRÖSSER GLEICH KLEINER
13876  7828
13877  7828
13878  7828  a6 4f      endrel ldx opmask      vom akt. Operator erzeugte Bit-Maske
13879  782a  d0 2c            bne finrel      GRÖSSER GLEICH KLEINER
13880  782c
13881  782c  ===> Test auf Funktionstoken <===
13882  782c
13883  782c  b0 7e            bcs qop         Token > $b3 (Funktions Token)
13884  782e
13885  782e  ===> Test auf PLUS, MINUS, MULT., DIV., POT., AND, OR <===
13886  782e
13887  782e  69 07            adc #_great-_plust
13888  7830  90 7a            bcc qop         nicht in diesem Bereich
13889  7832
13890  7832  ===> Test auf PLUS mit String <===
13891  7832
13892  7832  65 0f            adc valtyp      PLUS=1, String=$ff
13893  7834  d0 03            bne priopt      Zeiger auf Prioritäts Flag berechnen
13894  7836
13895  7836  ===> PLUS mit String <===
13896  7836
13897  7836  4c 0d 87         jmp cat         String verketten (+)
13898  7839
13899  7839
13900  7839  ===> Zeiger auf Prioritäts Flag berechnen <===
13901  7839
13902  7839  69 ff      priopt adc #$ff       minus 1
13903  783b
13904  783b  AC: 1 = PLUS
13905  783b      2 = MINUS
13906  783b      3 = MULT.
13907  783b      4 = DIV.
13908  783b      5 = POT.
13909  783b      6 = AND
13910  783b      7 = OR
13911  783b      8 = NEG
13912  783b      9 = NOT
13913  783b     10 = <=>
13914  783b
13915  783b  85 24            sta index1      ein mal
13916  783d
13917  783d  0a               asl a           mal zwei
13918  783e  65 24            adc index1      plus ein mal = mal drei
13919  7840  a8               tay             Zeiger auf akt. Prioritäts Flag
13920  7841
13921  7841  68         qprec  pla            altes Prioritäts Flag
```

```
13922  7842
13923  7842  d9 28 48          cmp optab,y       akt. Prioritäts Flag
13924  7845  b0 6a             bcs qchnum        altes größer/gleich
13925  7847
13926  7847  ===> akt. PRIO Flag größer, Funktion aufrufen <===
13927  7847
13928  7847  20 da 77          jsr chknum        Test num. VAR, sonst type mism.
13929  784a
13930  784a  48          doprec pha
13931  784b  20 71 78    negprc jsr dopre1       Routinen-Aufruf
13932  784e  68                 pla
13933  784f
13934  784f  a4 4d             ldy opptr         Zeiger auf PRIO Flag
13935  7851  10 17             bpl qprec1
13936  7853
13937  7853  aa                 tax               Prioritäts Flag
13938  7854  f0 59             beq qopgo         Ende Auswertung
13939  7856  d0 62             bne pulstk        ARG vom Stapel
13940  7858
13941  7858
13942  7858  46 0f       finrel lsr valtyp       SEC für String
13943  785a
13944  785a   $00 = numerisch
13945  785a   $ff = String
13946  785a
13947  785a  8a                 txa               Operator Maske
13948  785b  2a                 rol a             mal zwei, SET Bit0 für String
13949  785c
13950  785c  a6 3d              ldx txtptr        ADDRL chrget-Zeiger
13951  785e  d0 02              bne finre2
13952  7860  c6 3e              dec txtptr+1      ADDRH chrget-Zeiger
13953  7862  c6 3d       finre2 dec txtptr        ADDRL chrget-Zeiger
13954  7864
13955  7864  a0 1b              ldy #27           Zeiger auf PRIO Flag < = >
13956  7866
13957  7866  85 4f             sta opmask        vom akt. Operator erzeugte Bit-Maske
13958  7868
13959  7868   7654....  nicht benutzt
13960  7868   ....3...  1 = KLEINER (8)
13961  7868   .....2..  1 = GLEICH  (4)
13962  7868   ......1.  1 = GRÖSSER (2)
13963  7868   .......0  1 = String  (1)
13964  7868
13965  7868  d0 d7              bne qprec         immer
13966  786a
13967  786a
13968  786a  d9 28 48    qprec1 cmp optab,y       PRIO Flag
13969  786d  b0 4b              bcs pulstk
13970  786f  90 d9              bcc doprec
13971  7871
13972  7871
13973  7871  ===> alle Fkt. Parameter auf Stapel <===
13974  7871
13975  7871  b9 2a 48    dopre1 lda optab+2,y     ADDRH Funktions Routine
13976  7874  48                 pha
13977  7875
13978  7875  b9 29 48           lda optab+1,y     ADDRL Funktions Routine
13979  7878  48                 pha
```

```
13980  7879
13981  7879  20 84 78          jsr pushf        FAC auf Stapel
13982  787c
13983  787c  ********** BUG: PUSHF als JSR überflüssig ************
13984  787c
13985  787c  a5 4f             lda opmask       vom akt. Operator erzeugte Bit-Maske
13986  787e  4c fa 77          jmp lpoper       Auswerte Schleife
13987  7881
13988  7881
13989  7881  ===> Behandlung »?syntax error« <===
13990  7881
13991  7881  4c 6c 79    snerr5 jmp snerr      Behandlung »?syntax error«
13992  7884
13993  7884
13994  7884  ===> FAC auf Stapel <===
13995  7884
13996  7884  a5 68       pushf  lda facsgn     FAC: SG
13997  7886  be 28 48           ldx optab,y    akt. PRIO Flag (auf Stapel lpoper)
13998  7889  a8                 tay            FAC: SG
13999  788a
14000  788a  ===> RTS Adresse vom Stapel, nach index1 für ind. RTS <===
14001  788a
14002  788a  18                 clc
14003  788b  68                 pla            ADDRL Rücksprung-1
14004  788c  69 01              adc #1
14005  788e  85 24              sta index1     ADDRL Rücksprung
14006  7890
14007  7890  68                 pla            ADDRH Rücksprung-1
14008  7891  69 00              adc #0
14009  7893  85 25              sta index1+1   ADDRH Rücksprung
14010  7895
14011  7895  98                 tya            FAC: SG
14012  7896  48                 pha
14013  7897
14014  7897  20 47 8c           jsr round      FAC runden
14015  789a
14016  789a  a5 67              lda faclo      FAC: M1
14017  789c  48                 pha
14018  789d
14019  789d  a5 66              lda facmo      FAC: M2
14020  789f  48                 pha
14021  78a0
14022  78a0  a5 65              lda facmoh     FAC: M3
14023  78a2  48                 pha
14024  78a3
14025  78a3  a5 64              lda facho      FAC: M4
14026  78a5  48                 pha
14027  78a6
14028  78a6  a5 63              lda facexp     FAC: EX
14029  78a8  48                 pha
14030  78a9
14031  78a9  6c 24 00           jmp (index1)   Rücksprung statt RTS
14032  78ac
14033  78ac
14034  78ac  a0 ff       qop    ldy #$ff       Flag opptr
14035  78ae
14036  78ae  68                 pla            altes PRIO Flag
14037  78af
```

```
14038  78af  f0 23      qopgo  beq qoprts       Ende Auswertung
14039  78b1
14040  78b1  c9 64      qchnum cmp #$64          PRIO Flag GRÖSSER GLEICH KLEINER
14041  78b3  f0 03             beq unpstk       erlaubt für String und numerisch
14042  78b5
14043  78b5  20 da 77           jsr chknum       Test num. VAR, sonst type mism.
14044  78b8
14045  78b8  84 4d      unpstk sty opptr         Zeiger auf PRIO Flag
14046  78ba
14047  78ba  68         pulstk pla               FAC vom Stapel nach ARG
14048  78bb  4a                lsr a
14049  78bc  85 14             sta tansgn       Flag für TAN-Vorzeichen
14050  78be
14051  78be  68                pla
14052  78bf  85 6a             sta argexp       ARG: EX
14053  78c1
14054  78c1  68                pla
14055  78c2  85 6b             sta argho        ARG: M4
14056  78c4
14057  78c4  68                pla
14058  78c5  85 6c             sta argmoh       ARG: M3
14059  78c7
14060  78c7  68                pla
14061  78c8  85 6d             sta argmo        ARG: M2
14062  78ca
14063  78ca  68                pla
14064  78cb  85 6e             sta arglo        ARG: M1
14065  78cd
14066  78cd  68                pla
14067  78ce  85 6f             sta argsgn       ARG: SG
14068  78d0
14069  78d0  45 68             eor facsgn       FAC: SG
14070  78d2  85 70             sta arisgn       Vorzeichen Ergebnis Operation
14071  78d4
14072  78d4
14073  78d4  ===> Ende Auswertung <===
14074  78d4
14075  78d4  a5 63      qoprts lda facexp        FAC: EX
14076  78d6  60               rts
14077  78d7
14078  78d7
14079  78d7  ===> ind. folg. Ausdruck auswerten <===
14080  78d7
14081  78d7  6c 0a 03   eval   jmp (ieval)       ind. folg. Ausdruck auswerten
                                                ($78da)
14082  78da
14083  78da
14084  78da  ===> folg. Ausdruck auswerten <===
14085  78da
14086  78da  a9 00      neval  lda #_numer
14087  78dc  85 0f             sta valtyp       Flag Variablentyp
14088  78de
14089  78de  20 80 03   eval0  jsr chrget       nächstes Zeichen nach ac
14090  78e1  b0 05             bcs eval2        Test Ziffer/Var.name
14091  78e3
14092  78e3
```

```
14093  78e3  ===> numer. Wert nach FAC <===
14094  78e3
14095  78e3  a2 00      eval1  ldx #_pgmbk      PGM BANK
14096  78e5  4c 22 8d          jmp fin          Zahlenstr. (Bnk.=xr) nach FAC
14097  78e8
14098  78e8
14099  78e8
14099  78e8  ===> Test Ziffer/Var.name <===
14100  78e8
14101  78e8  20 3c 7b   eval2  jsr isletc       Test ac auf ALPHA (CARRY=1)
14102  78eb  90 03            bcc eval5
14103  78ed
14104  78ed  ===> Variable auswerten <===
14105  78ed
14106  78ed  4c 78 79          jmp isvar        Variable auswerten
14107  78f0
14108  78f0
14109  78f0  c9 ff      eval5  cmp #_pi         Code für PI
14110  78f2  d0 0f            bne qdot
14111  78f4
14112  78f4  ===> GK-Konstante PI nach FAC <===
14113  78f4
14114  78f4  a9 fe            lda #<pival      GK-Konstante PI (3.14159265)
14115  78f6  a0 78            ldy #>pival      GK-Konstante PI (3.14159265)
14116  78f8  20 d4 8b         jsr movfm        RAM Var. ac/yr nach FAC
14117  78fb
14118  78fb  4c 80 03         jmp chrget       nächstes Zeichen nach ac
14119  78fe
14120  78fe
14121  78fe  ===> GK-Konstante PI (3.14159265) <===
14122  78fe
14123  78fe  82 49 0f   pival  .byte $82, $49, $0f, $da, $a1
14123  7901  da a1
14124  7903
14125  7903
14126  7903  c9 2e      qdot   cmp #'.'
14127  7905  f0 dc            beq eval1        numer. Wert nach FAC
14128  7907
14129  7907  c9 ab            cmp #_minut      MINUS Token
14130  7909  f0 66            beq domin        Bearbeitung MINUS
14131  790b
14132  790b  c9 aa            cmp #_plust      PLUS Token
14133  790d  f0 cf            beq eval0
14134  790f
14135  790f  c9 22            cmp #'"'
14136  7911  d0 15            bne eval3
14137  7913
14138  7913
14139  7913  ===> String aus PGM nach Arb.string <===
14140  7913
14141  7913  a5 3d      strtxt lda txtptr       ADDRL chrget-Zeiger
14142  7915  a4 3e            ldy txtptr+1     ADDRH chrget-Zeiger
14143  7917  69 00            adc #0           hinter ""
14144  7919  90 01            bcc strtx2
14145  791b  c8                iny
14146  791c
14147  791c  20 9a 86   strtx2 jsr strlit       String (Descr.ac/yr) nach Arb.string
14148  791f
```

```
14149  791f
14150  791f  ===> txtptr hinter String in PGM <===
14151  791f
14152  791f  a6 72      st2txt ldx fbufpt       ADDRL Zeiger in Puffer
14153  7921  a4 73             ldy fbufpt+1     ADDRH Zeiger in Puffer
14154  7923
14155  7923  86 3d             stx txtptr       ADDRL chrget-Zeiger
14156  7925  84 3e             sty txtptr+1     ADDRH chrget-Zeiger
14157  7927
14158  7927  60                 rts
14159  7928
14160  7928
14161  7928  c9 a8      eval3  cmp #_nott       NOT Token
14162  792a  d0 16             bne eval4
14163  792c
14164  792c  a0 18             ldy #24          Zeiger auf PRIO Flag NOT
14165  792e  d0 43             bne gonprc       immer
14166  7930
14167  7930
14168  7930  ===> BASIC-Funktion NOT ($a8) <===
14169  7930
14170  7930  20 b4 84   notop  jsr ayint        Wandlung FAC nach sgnd. INT
14171  7933
14172  7933  a5 67             lda faclo        FAC: M1
14173  7935  49 ff             eor #%11111111
14174  7937  a8                 tay
14175  7938
14176  7938  a5 66             lda facmo        FAC: M2
14177  793a  49 ff             eor #%11111111
14178  793c
14179  793c
14180  793c  ===> 16 Bit INT yr/ac nach GK in FAC <===
14181  793c
14182  793c  20 e5 84   givayv jsr stoint       16 Bit INT yr/ac nach FAC
14183  793f  4c 70 8c          jmp floats       8/16 Bit EX nach FAC, normal.
14184  7942
14185  7942
14186  7942  c9 a5      eval4  cmp #_fnt        FN Token
14187  7944  d0 03             bne eval6
14188  7946
14189  7946  ===> FN auswerten <===
14190  7946
14191  7946  4c 3b 85          jmp fndoev       FN auswerten
14192  7949
14193  7949
14194  7949  c9 b4      eval6  cmp #_onefn      niedrigster Funktions Token
14195  794b  90 03             bcc parchk       Test auf folg.»(' ')«, sonst Fehler
14196  794d
14197  794d  ===> BASIC Funktion bearbeiten <===
14198  794d
14199  794d  4c f7 4b          jmp isfun        BASIC-Funktionen bearbeiten
14200  7950
14201  7950
14202  7950  ===> Test auf folg.»(' ')«, sonst Fehler <===
14203  7950
14204  7950  20 59 79   parchk jsr chkopn       Test auf folg. »(«, sonst Fehler
14205  7953  20 ef 77          jsr frmevl       folgenden Ausdruck auswerten
14206  7956
```

```
14207  7956
14208  7956  ===> Test auf folg. »)«, sonst Fehler <===
14209  7956
14210  7956  a9 29      chkcls lda #')'
14211  7958  2c                .byte _skip2
14212  7959
14213  7959
14214  7959
14214  7959  ===> Test auf folg. »(«, sonst Fehler <===
14215  7959
14216  7959  a9 28      chkopn lda #'('
14217  795b  2c                .byte _skip2
14218  795c
14219  795c
14220  795c  ===> Test auf folg. »Komma« sonst Fehler <===
14221  795c
14222  795c  a9 2c      chkcom lda #','
14223  795e
14224  795e
14225  795e  ===> Test auf Zeichen in ac, sonst Fehler <===
14226  795e
14227  795e  a0 00      synchr ldy #0
14228  7960  85 79             sta syntmp       ac aus chrget
14229  7962  20 c9 03          jsr indtxt       LDA (txtptr),y aus PGM-RAM
14230  7965  c5 79             cmp syntmp       ac aus chrget
14231  7967  d0 03             bne snerr        Behandlung »?syntax error«
14232  7969
14233  7969  4c 80 03          jmp chrget       nächstes Zeichen nach ac
14234  796c
14235  796c
14236  796c  ===> Behandlung »?syntax error« <===
14237  796c
14238  796c  a2 0b      snerr  ldx #_ersn
14239  796e  4c 3c 4d          jmp error        ind. Behandlung Fehlermeldung
14240  7971
14241  7971
14242  7971  ===> Bearbeitung MINUS <===
14243  7971
14244  7971  a0 15      domin  ldy #21          Zeiger auf PRIO Flag NEG. VORZ.
14245  7973
14246  7973  68         gonprc pla             Rücksprung Adresse
14247  7974  68                pla             verwerfen
14248  7975  4c 4b 78          jmp negprc
14249  7978
14250  7978
14251  7978  ===> Variable auswerten <===
14252  7978
14253  7978  20 af 7a   isvar  jsr ptrget       folg. Variable suchen/anlegen
14254  797b  85 66             sta indice       ADDRL Variable
14255  797d  84 67             sty indice+1     ADDRH Variable
14256  797f
14257  797f  a6 47             ldx varnam       erstes Zeichen akt. Variablenname
14258  7981  a4 48             ldy varnam+1     zweites Zeichen akt. Variablenname
14259  7983
14260  7983  a5 0f             lda valtyp       Flag Variablentyp
14261  7985  f0 63             beq isnum        num. Variable nach FAC
14262  7987
```

```
14263  7987  ===> Test auf System String Variable <===
14264  7987
14265  7987  a9 00           lda #0
14266  7989  85 71           sta facov        Rundungsbyte
14267  798b
14268  798b  ===> Test auf TI$ <===
14269  798b
14270  798b  e0 54           cpx #'t'
14271  798d  d0 25           bne isvds        Test auf DS$
14272  798f
14273  798f  c0 c9           cpy #'I'
14274  7991  d0 20           bne strrt1
14275  7993
14276  7993  ===> Test Zuweisung Systemzeit an TI$ String <===
14277  7993
14278  7993  a5 66           lda indice       ADDRL Variable
14279  7995  c9 d2           cmp #<zero       Konstante Null
14280  7997  d0 1a           bne strrt1
14281  7999
14282  7999  a5 67           lda indice+1     ADDRH Variable
14283  799b  c9 03           cmp #>zero       Konstante Null
14284  799d  d0 14           bne strrt1
14285  799f
14286  799f  ===> Zuweisung Systemzeit an TI$ String <===
14287  799f
14288  799f  20 1a 7a        jsr gettim       System Zeit nach FAC
14289  79a2  84 60           sty tenexp       0
14290  79a4
14291  79a4  88              dey              $ff
14292  79a5  84 72           sty fbufpt       ADDRL Zeiger in Puffer
14293  79a7
14294  79a7  a0 06           ldy #6
14295  79a9  84 5f           sty deccnt
14296  79ab
14297  79ab  a0 24           ldy #timcon-foutbl
14298  79ad  20 cd 8e        jsr foutim
14299  79b0  4c b8 85        jmp timstr
14300  79b3
14301  79b3
14302  79b3  60        strrt1 rts
14303  79b4
14304  79b4
14305  79b4  ===> Test auf DS$ <===
14306  79b4
14307  79b4  e0 44     isvds  cpx #'d'
14308  79b6  d0 fb            bne strrt1
14309  79b8
14310  79b8  c0 d3            cpy #'S'
14311  79ba  d0 f7            bne strrt1
14312  79bc
14313  79bc  ===> Zuweisung Disk Status an DS$ String <===
14314  79bc
14315  79bc  20 e3 79         jsr chkds        Test / Disk Status lesen
14316  79bf
14317  79bf  ===> tats. Länge Disk Status String nach yr <===
14318  79bf
14319  79bf  a0 ff            ldy #$ff
14320  79c1  c8        isds1  iny
```

```
14321  79c2
14322  79c2  a9 7b           lda #<dsdesc+1   Desc. DS$ (ADDRL)
14323  79c4  20 ab 03        jsr iram1        LDA (ac),y aus VAR-RAM
14324  79c7
14325  79c7  c9 00           cmp #_eom
14326  79c9  d0 f6           bne isds1
14327  79cb
14328  79cb  98              tya              tats. Länge DS$ String
14329  79cc  20 88 86        jsr strini       Arbeitsstring Länge ac anlegen
14330  79cf
14331  79cf  a8              tay              Länge Arbeitsstring
14332  79d0  f0 0e           beq isds0
14333  79d2
14334  79d2
14335  79d2  ===> Disk Status nach Arb.string <===
14336  79d2
14337  79d2  88        cpyds dey
14338  79d3  a9 7b           lda #<dsdesc+1   Desc. DS$ (ADDRL)
14339  79d5  20 ab 03        jsr iram1        LDA (ac),y aus VAR-RAM
14340  79d8  91 37           sta (frespc),y   ADDRL Zeiger auf neuen String
14341  79da
14342  79da  98              tya
14343  79db  d0 f5           bne cpyds        Disk Status nach Arb.string
14344  79dd
14345  79dd  20 71 87        jsr mvdone       ac zu frespc addieren
14346  79e0
14347  79e0  ********** ?BUG? ac=0 **************
14348  79e0
14349  79e0  4c e3 86   isds0 jmp putnew       String-Descriptor in Tabelle
14350  79e3
14351  79e3
14352  79e3  ===> Test / Disk Status lesen <===
14353  79e3
14354  79e3  a5 7a      chkds lda dsdesc       Desc. DS$ (Länge)
14355  79e5  d0 40           bne strrts       DS$ String gültig
14356  79e7
14357  79e7  4c 78 a7        jmp errchl       DS$ über Sek.adr. 15 einlesen
14358  79ea
14359  79ea
14360  79ea  ===> num. Variable nach FAC <===
14361  79ea
14362  79ea  24 10      isnum bit intflg       Flag für Gleitkomma/Integer
14363  79ec  10 0f           bpl isflot       GK-Variable nach FAC
14364  79ee
14365  79ee  ===> 16 Bit INT nach FAC <===
14366  79ee
14367  79ee  a0 00           ldy #0
14368  79f0  20 e7 42        jsr indfmo       LDA (facmo),y aus VAR-RAM
14369  79f3  aa              tax              High-Byte INT
14370  79f4
14371  79f4  c8              iny
14372  79f5  20 e7 42        jsr indfmo       LDA (facmo),y aus VAR-RAM
14373  79f8
14374  79f8  a8              tay              Low-Byte INT
14375  79f9  8a              txa              High-Byte INT
14376  79fa  4c 3c 79        jmp givayv       16 Bit INT yr/ac nach GK in FAC
14377  79fd
14378  79fd
```

```
14379  79fd  ===> GK-Variable nach FAC <===
14380  79fd
14381  79fd  a5 67      isflot lda indice+1     ADDRH Variable
14382  79ff  c9 03             cmp #>zero       Konstante Null
14383  7a01  d0 7e             bne gomovf       GK Var. nach FAC
14384  7a03
14385  7a03  a5 66             lda indice       ADDRL Variable
14386  7a05  c9 d2             cmp #<zero       Konstante Null
14387  7a07  d0 78             bne gomovf       GK Var. nach FAC
14388  7a09
14389  7a09
14389  7a09  ===> Test auf TI <===
14390  7a09
14391  7a09  e0 54             cpx #'t'
14392  7a0b  d0 1b             bne qstatv       Test auf System Status Var. ST
14393  7a0d
14394  7a0d  c0 49             cpy #'i'
14395  7a0f  d0 70             bne gomovf       GK Var. nach FAC
14396  7a11
14397  7a11  ===> TI nach FAC <===
14398  7a11
14399  7a11  20 1a 7a          jsr gettim       System Zeit nach FAC
14400  7a14  98                tya              SG / Rundungsbyte
14401  7a15  a2 a0             ldx #_expcn+31
14402  7a17  4c 7b 8c          jmp floatb       xr nach EX, ac nach SG, FAC normal.
14403  7a1a
14404  7a1a
14405  7a1a  ===> System Zeit nach FAC <===
14406  7a1a
14407  7a1a  20 de ff   gettim jsr krdtim       System-Uhr nach ac, xr, yr
                                                (Sprungv.)
14408  7a1d  86 66             stx facmo        FAC: M2
14409  7a1f  84 65             sty facmoh       FAC: M3
14410  7a21  85 67             sta faclo        FAC: M1
14411  7a23
14412  7a23  a0 00             ldy #0
14413  7a25  84 64             sty facho        FAC: M4
14414  7a27
14415  7a27  60         strrts rts
14416  7a28
14417  7a28
14418  7a28  ===> Test auf System Status Var. ST <===
14419  7a28
14420  7a28  e0 53      qstatv cpx #'s'
14421  7a2a  d0 0a             bne qdsav        Test auf Disk Status Var. DS
14422  7a2c
14423  7a2c  c0 54             cpy #'t'
14424  7a2e  d0 51             bne gomovf       GK Var. nach FAC
14425  7a30
14426  7a30  ===> ST nach FAC <===
14427  7a30
14428  7a30  20 b7 ff          jsr kreads       RS-/System-Status nach ac (Sprungv.)
14429  7a33  4c 68 8c          jmp float        8 Bit INT ac nach FAC
14430  7a36
14431  7a36
14432  7a36  ===> Test auf Disk Status Var. DS <===
14433  7a36
14434  7a36  e0 44      qdsav  cpx #'d'
```

```
14435  7a38  d0 26           bne qerlin       Test Var. Fehlernr. ER / Fehlerz. EL
14436  7a3a
14437  7a3a  c0 53           cpy #'s'
14438  7a3c  d0 43           bne gomovf       GK Var. nach FAC
14439  7a3e
14440  7a3e  ===> DS nach FAC <===
14441  7a3e
14442  7a3e  20 e3 79        jsr chkds        Test / Disk Status lesen
14443  7a41
14444  7a41  a0 00           ldy #0           Zeiger 10er Stelle Fehlernr.
14445  7a43  a9 7b           lda #<dsdesc+1   Desc. DS$ (ADDRL)
14446  7a45
14447  7a45  ===> 10er Ziffer ASCII/HEX mal zehn <===
14448  7a45
14449  7a45  20 ab 03        jsr iram1        LDA (ac),y aus VAR-RAM
14450  7a48  29 0f           and #%00001111   ASCII nach HEX
14451  7a4a  0a              asl a            mal zwei
14452  7a4b  85 11           sta dores        Flag für DATA, LIST, garbage
                                             collection
14453  7a4d
14454  7a4d  0a              asl a
14455  7a4e  0a              asl a            mal acht
14456  7a4f  65 11           adc dores        mal zwei plus mal acht = mal zehn
14457  7a51  85 11           sta dores        10er Ziffer DS in HEX
14458  7a53
14459  7a53  ===> 1er Ziffer ASCII/HEX plus 10er / nach FAC <===
14460  7a53
14461  7a53  c8              iny              Zeiger 1er Stelle Fehlernr.
14462  7a54  a9 7b           lda #<dsdesc+1   Desc. DS$ (ADDRL)
14463  7a56  20 ab 03        jsr iram1        LDA (ac),y aus VAR-RAM
14464  7a59  29 0f           and #%00001111   ASCII nach HEX
14465  7a5b  65 11           adc dores        10er plus 1er = DS in HEX
14466  7a5d  4c 68 8c        jmp float        8 Bit INT ac nach FAC
14467  7a60
14468  7a60
14469  7a60  ===> Test Var. Fehlernr. ER / Fehlerz. EL <===
14470  7a60
14471  7a60  e0 45     qerlin cpx #'e'
14472  7a62  d0 1d           bne gomovf       GK Var. nach FAC
14473  7a64
14474  7a64  c0 52           cpy #'r'
14475  7a66  f0 10           beq qnumer       ER nach FAC
14476  7a68
14477  7a68  c0 4c           cpy #'l'
14478  7a6a  d0 15           bne gomovf       GK Var. nach FAC
14479  7a6c
14480  7a6c  ===> EL nach FAC <===
14481  7a6c
14482  7a6c  8d 03 ff        sta lcrc         ROM / PGM-RAM einschalten
14483  7a6f
14484  7a6f  ad 0a 12        lda errlin+1     High-Byte Zeilennummer für RESUME
14485  7a72  ac 09 12        ldy errlin       Low-Byte Zeilennummer für RESUME
14486  7a75  4c c9 84        jmp nosflt       16 Bit INT yr/ac nach GK in FAC
14487  7a78
14488  7a78
14489  7a78  ===> ER nach FAC <===
14490  7a78
14491  7a78  8d 03 ff  qnumer sta lcrc        ROM / PGM-RAM einschalten
```

```
14492  7a7b
14493  7a7b  ad 08 12          lda errnum        Nummer Fehlertext
14494  7a7e  4c 68 8c          jmp float         8 Bit INT ac nach FAC
14495  7a81
14496  7a81
14497  7a81  ===> GK Var. nach FAC <===
14498  7a81
14499  7a81  a5 66       gomovf lda indice        ADDRL Variable
14500  7a83  a4 67              ldy indice+1      ADDRH Variable
14501  7a85
14502  7a85  85 24       movfrm sta index1
14503  7a87  84 25              sty index1+1
14504  7a89
14505  7a89  a0 00              ldy #0            Zeiger auf EX
14506  7a8b  20 b7 03           jsr ind1_1        LDA (index1),y aus VAR-RAM
14507  7a8e  85 63              sta facexp        FAC: EX
14508  7a90
14509  7a90  84 71              sty facov         Rundungsbyte
14510  7a92
14511  7a92  c8                 iny               Zeiger auf M4
14512  7a93  20 b7 03           jsr ind1_1        LDA (index1),y aus VAR-RAM
14513  7a96  85 68              sta facsgn        FAC: SG
14514  7a98
14515  7a98  09 80              ora #%10000000    Bit7 restaurieren
14516  7a9a  85 64              sta facho         FAC: M4
14517  7a9c
14518  7a9c  c8                 iny               Zeiger auf M3
14519  7a9d  20 b7 03           jsr ind1_1        LDA (index1),y aus VAR-RAM
14520  7aa0  85 65              sta facmoh        FAC: M3
14521  7aa2
14522  7aa2  c8                 iny               Zeiger auf M2
14523  7aa3  20 b7 03           jsr ind1_1        LDA (index1),y aus VAR-RAM
14524  7aa6  85 66              sta facmo         FAC: M2
14525  7aa8
14526  7aa8  c8                 iny               Zeiger auf M1
14527  7aa9  20 b7 03           jsr ind1_1        LDA (index1),y aus VAR-RAM
14528  7aac  85 67              sta faclo         FAC: M1
14529  7aae
14530  7aae  60                 rts
14531  7aaf
14532  7aaf
14533  7aaf  ===> folg. Variable suchen/anlegen <===
14534  7aaf
14535  7aaf  a2 00       ptrget ldx #0            Flag Zeigr Var. suchen
14536  7ab1  20 86 03           jsr chrgot        letztes Zeichen erneut nach ac
14537  7ab4
14538  7ab4
14539  7ab4  ===> folg. VAR dim. / anlegen / Zeiger such. <===
14540  7ab4
14541  7ab4  86 0e       ptrgt1 stx dimflg        Flag DIM/Zeiger suchen
14542  7ab6
14543  7ab6
14544  7ab6  ===> Variable suchen oder anlegen <===
14545  7ab6
14546  7ab6  85 47       ptrgt2 sta varnam        erstes Zeichen akt. Variablenname
14547  7ab8
```

```
14548  7ab8  ===> Test 1. Zeichen VAR NAM numerisch <===
14549  7ab8
14550  7ab8  20 86 03          jsr chrgot       1. Zeichen VAR NAM
14551  7abb  20 3c 7b          jsr isletc       Test ac auf ALPHA (CARRY=1)
14552  7abe  b0 03             bcs ptrgt3
14553  7ac0
14554  7ac0  ===> Behandlung »?syntax error« <===
14555  7ac0
14556  7ac0  4c 6c 79   interr jmp snerr        Behandlung »?syntax error«
14557  7ac3
14558  7ac3
14559  7ac3  a2 00      ptrgt3 ldx #0
14560  7ac5  86 0f             stx valtyp       Variablentyp numerisch
14561  7ac7  86 10             stx intflg       INT VAR zugelassen
14562  7ac9
14563  7ac9
14563  7ac9  ===> dez.Ziff./Alphaz. Rest VAR NAM überlesen <===
14564  7ac9
14565  7ac9  20 80 03          jsr chrget       nächstes Zeichen nach ac
14566  7acc  90 05             bcc issec        dez.Ziffer
14567  7ace
14568  7ace  20 3c 7b          jsr isletc       Test ac auf ALPHA (CARRY=1)
14569  7ad1  90 0b             bcc nosec        Alphazeich.
14570  7ad3
14571  7ad3  aa         issec  tax              2. Zeichen VAR NAM
14572  7ad4
14573  7ad4
14574  7ad4  ===> weiter VAR NAM Rest überlesen <===
14575  7ad4
14576  7ad4  20 80 03   eatem  jsr chrget       nächstes Zeichen nach ac
14577  7ad7  90 fb             bcc eatem        Ziffer, weiter überlesen
14578  7ad9
14579  7ad9  20 3c 7b          jsr isletc       Test ac auf ALPHA (CARRY=1)
14580  7adc  b0 f6             bcs eatem        Alphaz., weiter überlesen
14581  7ade
14582  7ade  c9 24      nosec  cmp #'$'
14583  7ae0  d0 06             bne notstr       Test INT VAR
14584  7ae2
14585  7ae2  ===> String Variable <===
14586  7ae2
14587  7ae2  a9 ff             lda #$ff
14588  7ae4  85 0f             sta valtyp       Flag Variablentyp
14589  7ae6  d0 10             bne turnon       immer
14590  7ae8
14591  7ae8
14592  7ae8  ===> Test INT VAR <===
14593  7ae8
14594  7ae8  c9 25      notstr cmp #'%'
14595  7aea  d0 13             bne strnam
14596  7aec
14597  7aec  ===> Test / INT VAR <===
14598  7aec
14599  7aec  a5 12             lda sublfg       Flag Zulassen/Sperren INT-Var.
14600  7aee
14601  7aee   7.......  1 = INT nicht erlaubt
14602  7aee
14603  7aee  d0 d0             bne interr
14604  7af0
```

```
14605  7af0  ===> INT VAR <===
14606  7af0
14607  7af0  a9 80             lda #%10000000
14608  7af2  85 10             sta intflg       Flag für Gleitkomma/Integer
14609  7af4
14610  7af4  05 47             ora varnam       erstes Zeichen akt. Variablenname
14611  7af6  85 47             sta varnam       erstes Zeichen akt. Variablenname
14612  7af8
14613  7af8   INT VAR: in beiden Zeichen VAR NAM Bit7 gesetzt
14614  7af8
14615  7af8
14616  7af8  ===> Bit7 2. Zeichen VAR NAM setzen <===
14617  7af8
14618  7af8  8a          turnon txa             2. Zeichen VAR NAM
14619  7af9  09 80             ora #%10000000
14620  7afb  aa                tax              2. Zeichen VAR NAM
14621  7afc
14622  7afc  20 80 03          jsr chrget       nächstes Zeichen nach ac
14623  7aff
14624  7aff  86 48       strnam stx varnam+1     zweites Zeichen akt. Variablenname
14625  7b01
14626  7b01  38                sec
14627  7b02  05 12             ora sublfg       Integer-Array-Sperrflag
14628  7b04  e9 28             sbc #'('         Feld-Variable und zulässig?
14629  7b06  d0 03             bne strna1
14630  7b08
14631  7b08  4c ab 7c          jmp isary        ARY dim. / Zeiger auf ARY VAR
14632  7b0b
14633  7b0b
14634  7b0b  a0 00       strna1 ldy #$00
14635  7b0d  84 12             sty sublfg       Integer-Array-Sperrflag rücksetzen
14636  7b0f
14637  7b0f  a5 2f             lda vartab       ADDRL Zeiger Anfang einf. Variable
14638  7b11  a6 30             ldx vartab+1     ADDRH Zeiger Anfang einf. Variable
14639  7b13
14640  7b13
14641  7b13  ===> nächsten VAR NAM vergleichen <===
14642  7b13
14643  7b13  86 62       stxfnd stx lowtr+1      in Suchzeiger kopieren
14644  7b15
14645  7b15
14646  7b15  ===> Test Ende VAR TAB <===
14647  7b15
14648  7b15  85 61       lopfnd sta lowtr
14649  7b17
14650  7b17  e4 32             cpx arytab+1     ADDRH Zeiger Anfang Feld-Tabelle
14651  7b19  d0 04             bne lopfn
14652  7b1b
14653  7b1b  c5 31             cmp arytab       ADDRL Zeiger Anfang Feld-Tabelle
14654  7b1d  f0 27             beq notfns       VAR NAM nicht gefunden
14655  7b1f
14656  7b1f  20 00 43    lopfn  jsr indlo1       LDA (lowtr),y aus VAR-RAM
14657  7b22  c5 47             cmp varnam       erstes Zeichen akt. Variablenname
14658  7b24  d0 0c             bne notit
14659  7b26
14660  7b26  c8                iny
14661  7b27  20 00 43          jsr indlo1       LDA (lowtr),y aus VAR-RAM
14662  7b2a  c5 48             cmp varnam+1     zweites Zeichen akt. Variablenname
```

```
14663  7b2c  d0 03           bne nxtptr       Suchz. auf nächst. VAR Desc.
14664  7b2e
14665  7b2e  4c 57 7c        jmp finptr       Zeiger auf VAR/Desc. berechn.
14666  7b31
14667  7b31
14668  7b31  ===> Suchz. auf nächst. VAR Desc. <===
14669  7b31
14670  7b31  88         nxtptr dey
14671  7b32  18         notit  clc
14672  7b33  a5 61            lda lowtr
14673  7b35  69 07            adc #_dscln      Länge VAR Desc.
14674  7b37  90 dc            bcc lopfnd       Test Ende VAR TAB
14675  7b39  e8               inx
14676  7b3a  d0 d7            bne stxfnd       immer
14677  7b3c
14678  7b3c
14679  7b3c
14679  7b3c  ===> Test ac auf ALPHA (CARRY=1) <===
14680  7b3c
14681  7b3c  c9 41      isletc cmp #'a'
14682  7b3e  90 05            bcc islrts       CARRY = 0
14683  7b40
14684  7b40  e9 5b            sbc #'ä'         (Umlaute nicht Alphaz. !)
14685  7b42  38               sec
14686  7b43  e9 a5            sbc #$a5
14687  7b45
14688  7b45  60         islrts rts
14689  7b46
14690  7b46
14691  7b46  ===> VAR NAM nicht gefunden <===
14692  7b46
14693  7b46  ba         notfns tsx
14694  7b47  bd 02 01          lda stack+2,x    ADDRH Rücksprung JSR
14695  7b4a  c9 83            cmp #>pointr+13  BASIC-Funktion POINTER ($ce $0a)
14696  7b4c  f0 04            beq ldzr         RETURN Zeiger auf Konst. NULL
14697  7b4e
14698  7b4e  c9 79            cmp #>isvar+2    Variable auswerten
14699  7b50  d0 2a            bne notevl
14700  7b52
14701  7b52
14702  7b52  ===> RETURN Zeiger auf Konst. NULL <===
14703  7b52
14704  7b52  a9 d2      ldzr  lda #<zero       Konstante Null
14705  7b54  a0 03            ldy #>zero       Konstante Null
14706  7b56  60               rts
14707  7b57
14708  7b57
14709  7b57  ===> Test auf TI$ / TI <===
14710  7b57
14711  7b57  c0 c9      qst001 cpy #'I'
14712  7b59  f0 f7            beq ldzr         TI$: Zeiger auf NULL
14713  7b5b
14714  7b5b  c0 49            cpy #'i'
14715  7b5d  d0 31            bne varok        VAR NAM ok, ARY TAB verschieben
14716  7b5f  f0 18            beq gobadv       TI: Zuweisung nicht erlaubt
14717  7b61
14718  7b61
```

```
14719  7b61  ===> Test auf DS$ / DS <===
14720  7b61
14721  7b61  c0 d3      qst004 cpy #'S'
14722  7b63  f0 14             beq gobadv       DS$: Zuweisung nicht erlaubt
14723  7b65
14724  7b65  c0 53             cpy #'s'
14725  7b67  d0 27             bne varok        VAR NAM ok, ARY TAB verschieben
14726  7b69  f0 0e             beq gobadv       DS: Zuweisung nicht erlaubt
14727  7b6b
14728  7b6b
14729  7b6b  ===> Test auf ST <===
14730  7b6b
14731  7b6b  c0 54      qst002 cpy #'t'
14732  7b6d  d0 21             bne varok        VAR NAM ok, ARY TAB verschieben
14733  7b6f  f0 08             beq gobadv       ST: Zuweisung nicht erlaubt
14734  7b71
14735  7b71
14736  7b71
14736  7b71  ===> Test auf ER / EL <===
14737  7b71
14738  7b71  c0 52      qst003 cpy #'r'
14739  7b73  f0 04             beq gobadv       ER: Zuweisung nicht erlaubt
14740  7b75
14741  7b75  c0 4c             cpy #'l'
14742  7b77  d0 17             bne varok        VAR NAM ok, ARY TAB verschieben
14743  7b79
14744  7b79  ===> Behandlung »?syntax error« <===
14745  7b79
14746  7b79  4c 6c 79   gobadv jmp snerr        Behandlung »?syntax error«
14747  7b7c
14748  7b7c
14749  7b7c  a5 47      notevl lda varnam       erstes Zeichen akt. Variablenname
14750  7b7e  a4 48             ldy varnam+1     zweites Zeichen akt. Variablenname
14751  7b80
14752  7b80  c9 54             cmp #'t'
14753  7b82  f0 d3             beq qst001       Test auf TI$ / TI
14754  7b84
14755  7b84  c9 53             cmp #'s'
14756  7b86  f0 e3             beq qst002       Test auf ST
14757  7b88
14758  7b88  c9 45             cmp #'e'
14759  7b8a  f0 e5             beq qst003       Test auf ER / EL
14760  7b8c
14761  7b8c  c9 44             cmp #'d'
14762  7b8e  f0 d1             beq qst004       Test auf DS$ / DS
14763  7b90
14764  7b90
14765  7b90  ===> VAR NAM ok, ARY TAB verschieben <===
14766  7b90
14767  7b90  a5 31      varok  lda arytab       ADDRL Zeiger Anfang Feld-Tabelle
14768  7b92  a4 32             ldy arytab+1     ADDRH Zeiger Anfang Feld-Tabelle
14769  7b94
14770  7b94  85 61             sta lowtr        ADDRL Versch. QUELLE VON
14771  7b96  84 62             sty lowtr+1      ADDRH Versch. QUELLE VON
14772  7b98
14773  7b98  a5 33             lda strend       ADDRL+1 Obergr. ARRAY Tabelle
14774  7b9a  a4 34             ldy strend+1     ADDRH+1 Obergr. ARRAY Tabelle
14775  7b9c
```

```
14776  7b9c  85 5c           sta hightr       ADDRL Versch. QUELLE BIS
14777  7b9e  84 5d           sty hightr+1     ADDRH Versch. QUELLE BIS
14778  7ba0
14779  7ba0  18              clc
14780  7ba1  69 07           adc #_dscln      Länge VAR Desc.
14781  7ba3  90 01           bcc noteve
14782  7ba5  c8              iny
14783  7ba6
14784  7ba6  85 5a     noteve sta highds      ADDRL Versch. ZIEL BIS
14785  7ba8  84 5b           sty highds+1     ADDRH Versch. ZIEL BIS
14786  7baa
14787  7baa  20 66 7c        jsr bltu         Block in RAM verschieben
14788  7bad
14789  7bad  a5 5a           lda highds
14790  7baf  a4 5b           ldy highds+1
14791  7bb1
14792  7bb1  c8              iny
14793  7bb2  85 31           sta arytab       ADDRL Zeiger Anfang Feld-Tabelle
14794  7bb4  84 32           sty arytab+1     ADDRH Zeiger Anfang Feld-Tabelle
14795  7bb6
14796  7bb6  85 5a           sta highds
14797  7bb8  84 5b           sty highds+1
14798  7bba
14799  7bba
14800  7bba  ===> Trailer alle String ARY VAR korrigieren <===
14801  7bba
14802  7bba  a5 5a     aryva2 lda highds
14803  7bbc  a6 5b           ldx highds+1
14804  7bbe
14805  7bbe
14806  7bbe  ===> Test / nächstes ARY überprüfen <===
14807  7bbe
14808  7bbe  e4 34     aryva3 cpx strend+1     ADDRH+1 Obergr. ARRAY Tabelle
14809  7bc0  d0 06           bne aryvgo
14810  7bc2
14811  7bc2  c5 33           cmp strend       ADDRL+1 Obergr. ARRAY Tabelle
14812  7bc4  d0 02           bne aryvgo
14813  7bc6  f0 78           beq arydon       neue VAR in VAR TAB eintragen
14814  7bc8
14815  7bc8  85 24     aryvgo sta index1      ADDRL Kopf ARY VAR
14816  7bca  86 25           stx index1+1     ADDRH Kopf ARY VAR
14817  7bcc
14818  7bcc  a0 00           ldy #0           Zeiger auf 1. Zeichen VAR NAM
14819  7bce  20 b7 03        jsr ind1_1       LDA (index1),y aus VAR-RAM
14820  7bd1  aa              tax              1. Zeichen VAR NAM
14821  7bd2
14822  7bd2  c8              iny              Zeiger 2. Zeichen VAR NAM
14823  7bd3  20 b7 03        jsr ind1_1       LDA (index1),y aus VAR-RAM
14824  7bd6  08              php              Flag Bit7
14825  7bd7
14826  7bd7  c8              iny              Zeiger auf Low-Byte Länge ARY
14827  7bd8  20 b7 03        jsr ind1_1       LDA (index1),y aus VAR-RAM
14828  7bdb  65 5a           adc highds       ADDRL Anfang ARY
14829  7bdd  85 5a           sta highds       ADDRL Ende ARY
14830  7bdf
14831  7bdf  c8              iny              Zeiger auf High-Byte Länge ARY
14832  7be0  20 b7 03        jsr ind1_1       LDA (index1),y aus VAR-RAM
14833  7be3  65 5b           adc highds+1     ADDRH Anfang akt. ARY
```

```
14834  7be5  85 5b           sta highds+1     ADDRH Ende akt. ARY
14835  7be7
14836  7be7  28              plp              Status 2. Zeichen VAR NAM
14837  7be8  10 d0           bpl aryva2       GK ARY (ohne Trailer)
14838  7bea
14839  7bea  8a              txa              1. Zeichen VAR NAM
14840  7beb  30 cd           bmi aryva2       INT ARY (ohne Trailer)
14841  7bed
14842  7bed  ===> String ARRAY: Trailer korrigieren <===
14843  7bed
14844  7bed  c8              iny              Zeiger auf Anzahl DIM
14845  7bee  20 b7 03        jsr ind1_1       LDA (index1),y aus VAR-RAM
14846  7bf1
14847  7bf1  a0 00           ldy #0
14848  7bf3  0a              asl a            DIM mal zwei
14849  7bf4  69 05           adc #_aryhd      plus Länge ARY Kopf
14850  7bf6  65 24           adc index1       ADDRL Kopf ARY
14851  7bf8  85 24           sta index1       ADDRL 1. String Desc. ARY
14852  7bfa  90 02           bcc aryget       Test ARY Ende
14853  7bfc  e6 25           inc index1+1     ADDRH 1. String Desc. ARY
14854  7bfe
14855  7bfe
14856  7bfe  ===> Test ARY Ende <===
14857  7bfe
14858  7bfe  a6 25     aryget ldx index1+1     ADDRH String Desc. ARY
14859  7c00  e4 5b           cpx highds+1     ADDRH Ende akt. ARY
14860  7c02  d0 04           bne gogo         Trailer korrigieren
14861  7c04
14862  7c04  c5 5a           cmp highds       ADDRL Ende akt. ARY
14863  7c06  f0 b6           beq aryva3       Test / nächstes ARY überprüfen
14864  7c08
14865  7c08
14866  7c08  ===> Trailer korrigieren <===
14867  7c08
14868  7c08  a0 00      gogo ldy #0           Zeiger auf String Länge
14869  7c0a  20 b7 03        jsr ind1_1       LDA (index1),y aus VAR-RAM
14870  7c0d  f0 24           beq dvarts       Zeiger auf nächsten String Desc.
14871  7c0f
14872  7c0f  85 79           sta lentmp       Stringlänge
14873  7c11
14874  7c11  c8              iny              Zeiger auf ADDRL String
14875  7c12  20 b7 03        jsr ind1_1       LDA (index1),y aus VAR-RAM
14876  7c15  18              clc              ADDRL String
14877  7c16  65 79           adc lentmp
14878  7c18  85 5c           sta hightr       ADDRL Trailer
14879  7c1a
14880  7c1a  c8              iny              Zeiger auf ADDRH String
14881  7c1b  20 b7 03        jsr ind1_1       LDA (index1),y aus VAR-RAM
14882  7c1e  69 00           adc #0
14883  7c20  85 5d           sta hightr+1     ADDRH Trailer
14884  7c22
14885  7c22  ===> Trailer korrigieren <===
14886  7c22
14887  7c22  a0 00           ldy #0           Zeiger ADDRL Desc. in Trailer
14888  7c24  20 e2 42        jsr inhtr1       LDA (hightr),y aus VAR-RAM
14889  7c27  69 07           adc #_dscln
14890  7c29  91 5c           sta (hightr),y   ADDRL Desc. in Trailer
14891  7c2b
```

```
14892  7c2b  c8                iny             Zeiger ADDRH Desc. in Trailer
14893  7c2c  20 e2 42          jsr inhtr1      LDA (hightr),y aus VAR-RAM
14894  7c2f  69 00             adc #0
14895  7c31  91 5c             sta (hightr),y  ADDRH Desc. in Trailer
14896  7c33
14897  7c33
14898  7c33  ===> Zeiger auf nächsten String Desc. <===
14899  7c33
14900  7c33  a9 03       dvarts lda #_aryln    Länge String Descr. ARY
14901  7c35  18               clc
14902  7c36  65 24             adc index1      ADDRL String Desc. ARY
14903  7c38  85 24             sta index1      ADDRL nächster String Desc. ARY
14904  7c3a  90 c2             bcc aryget      Test ARY Ende
14905  7c3c
14906  7c3c  e6 25             inc index1+1    ADDRH nächster String Desc. ARY
14907  7c3e  d0 be             bne aryget      Test ARY Ende
14908  7c40
14909  7c40
14910  7c40
14910  7c40  ===> neue VAR in VAR TAB eintragen <===
14911  7c40
14912  7c40  a0 00       arydon ldy #0
14913  7c42  a5 47             lda varnam      erstes Zeichen akt. Variablenname
14914  7c44  8d 04 ff          sta lcrd        ROM / VAR-RAM einschalten
14915  7c47
14916  7c47  91 61             sta (lowtr),y   VAR Kopf
14917  7c49
14918  7c49  c8                iny
14919  7c4a  a5 48             lda varnam+1    zweites Zeichen akt. Variablenname
14920  7c4c  91 61             sta (lowtr),y   VAR Kopf
14921  7c4e
14922  7c4e  ===> VAR Wert rücksetzen <===
14923  7c4e
14924  7c4e  a9 00             lda #0
14925  7c50  c8          arydlp iny
14926  7c51  91 61             sta (lowtr),y   VAR Wert
14927  7c53
14928  7c53  c0 06             cpy #_dscln-1
14929  7c55  d0 f9             bne arydlp
14930  7c57
14931  7c57
14932  7c57  ===> Zeiger auf VAR/Desc. berechn. <===
14933  7c57
14934  7c57  a5 61       finptr lda lowtr       ADDRL VAR NAM
14935  7c59  18               clc
14936  7c5a  69 02             adc #2
14937  7c5c  a4 62             ldy lowtr+1
14938  7c5e  90 01             bcc finpt1      Zeiger auf VAR / Desc. nach varpnt
14939  7c60  c8                iny
14940  7c61
14941  7c61
14942  7c61  ===> Zeiger auf VAR / Desc. nach varpnt <===
14943  7c61
14944  7c61  85 49       finpt1 sta varpnt      ADDRL Zeiger auf Variable im RAM
14945  7c63  84 4a             sty varpnt+1    ADDRH Zeiger auf Variable im RAM
14946  7c65
14947  7c65  60                rts
14948  7c66
```

```
14949  7c66
14950  7c66                    .end
14951  7c66                    .lib basic.array
14952  7c66  put"@basic.array"
```

```
14954  7c66
14955  7c66
14956  7c66  ===> Block in RAM verschieben <===
14957  7c66
14958  7c66  20 17 50   bltu   jsr reason       Test Platz in VAR-RAM, sonst
                                                garb.coll.
14959  7c69
14960  7c69  85 33             sta strend      ADDRL+1 Obergr. ARRAY Tabelle
14961  7c6b  84 34             sty strend+1    ADDRH+1 Obergr. ARRAY Tabelle
14962  7c6d
14963  7c6d  ===> Anzahl (256 Byte) Blöcke und Block Rest berechnen <===
14964  7c6d
14965  7c6d  38                sec
14966  7c6e  a5 5c             lda hightr      ADDRL+1 BIS Quelle
14967  7c70  e5 61             sbc lowtr       ADDRL VON Quelle
14968  7c72
14969  7c72  85 24             sta index1      Block Rest
14970  7c74  a8                tay             Block Rest
14971  7c75
14972  7c75  a5 5d             lda hightr+1    ADDRH+1 BIS Quelle
14973  7c77  e5 62             sbc lowtr+1     ADDRH VON Quelle
14974  7c79
14975  7c79  aa                tax             Blöcke
14976  7c7a  e8                inx             Block Zähler
14977  7c7b
14978  7c7b  98                tya             Block Rest
14979  7c7c  f0 25             beq decblt      Block Rest = 0
14980  7c7e
14981  7c7e
14982  7c7e  ===> Zeiger auf Ende letzten Quell Block berechnen <===
14983  7c7e
14984  7c7e  a5 5c             lda hightr      ADDRL+1 BIS Quelle
14985  7c80  38                sec
14986  7c81  e5 24             sbc index1      Block Rest
14987  7c83  85 5c             sta hightr      ADDRL Ende letzter Quell Block
14988  7c85
14989  7c85  b0 03             bcs bltu1
14990  7c87  c6 5d             dec hightr+1    ADDRH Ende letzter Quell Block
14991  7c89  38                sec
14992  7c8a
14993  7c8a  ===> Zeiger auf Ende letzten Ziel Block berechnen <===
14994  7c8a
14995  7c8a  a5 5a      bltu1  lda highds      ADDRL+1 BIS Ziel
14996  7c8c  e5 24             sbc index1      Block Rest
14997  7c8e  85 5a             sta highds      ADDRL Ende letzter Ziel Block
14998  7c90
14999  7c90  b0 09             bcs moren1
15000  7c92  c6 5b             dec highds+1    ADDRH Ende letzter Ziel Block
15001  7c94  90 05             bcc moren1      Block Rest verschieben
15002  7c96
15003  7c96
15004  7c96  ===> Verschieben Quelle nach Ziel <===
15005  7c96
15006  7c96  20 e2 42   bltlp  jsr inhtr1      Quelle
15007  7c99  91 5a             sta (highds),y  Ziel
15008  7c9b
15009  7c9b  88         moren1 dey             Zeiger in Block
15010  7c9c  d0 f8             bne bltlp       Verschieben Quelle nach Ziel
```

```
15011  7c9e
15012  7c9e  20 e2 42          jsr inhtr1       Quelle Block letzt. Byte
15013  7ca1  91 5a             sta (highds),y   Ziel Block letzt. Byte
15014  7ca3
15015  7ca3  c6 5d      decblt dec hightr+1     ADDRH BIS Quelle
15016  7ca5  c6 5b             dec highds+1     ADDRH BIS Ziel
15017  7ca7  ca                dex              Block Zähler
15018  7ca8  d0 f1             bne moren1
15019  7caa
15020  7caa  60                rts
15021  7cab
15022  7cab
15023  7cab  ===> ARY dim. / Zeiger auf ARY VAR <===
15024  7cab
15025  7cab  a5 0e       isary  lda dimflg       Flag DIM/Zeiger suchen
15026  7cad
15027  7cad   01xxxxxx  Aufruf durch DIM (1. Zeich. VAR NAM)
15028  7cad   00000000  Aufruf durch ptrget (Zeiger suchen)
15029  7cad
15030  7cad  05 10             ora intflg       Flag für Gleitkomma/Integer
15031  7caf
15032  7caf   7.......  1 = INT VAR, 0 = GK VAR
15033  7caf
15034  7caf  48                pha              Flag DIM/SUCH, INT/GK
15035  7cb0
15036  7cb0  a5 0f             lda valtyp       Flag Variablentyp
15037  7cb2
15038  7cb2   0 = numerisch, 255 = String
15039  7cb2
15040  7cb2  48                pha
15041  7cb3
15042  7cb3  a0 00             ldy #0
15043  7cb5
15044  7cb5  98          indlop tya
15045  7cb6  48                pha
15046  7cb7
15047  7cb7  a5 48             lda varnam+1     zweites Zeichen akt. Variablenname
15048  7cb9  48                pha
15049  7cba  a5 47             lda varnam       erstes Zeichen akt. Variablenname
15050  7cbc  48                pha
15051  7cbd
15052  7cbd  20 a7 84          jsr intidx       folg. Variable als pos. INT nach FAC
15053  7cc0
15054  7cc0  68                pla
15055  7cc1  85 47             sta varnam       erstes Zeichen akt. Variablenname
15056  7cc3  68                pla
15057  7cc4  85 48             sta varnam+1     zweites Zeichen akt. Variablenname
15058  7cc6
15059  7cc6  68                pla
15060  7cc7  a8                tay
15061  7cc8
15062  7cc8  ===> INDEX ARY als 16 Bit INT auf Stapel übergeben <===
15063  7cc8
15064  7cc8  ba                tsx
15065  7cc9  bd 02 01          lda stack+2,x    dimflg or intflg
15066  7ccc  48                pha
15067  7ccd
15068  7ccd  bd 01 01          lda stack+1,x    valtyp
```

```
15069  7cd0  48              pha
15070  7cd1
15071  7cd1  a5 66           lda facmo       Low-Byte INDEX ARY
15072  7cd3  9d 02 01        sta stack+2,x
15073  7cd6
15074  7cd6  a5 67           lda faclo       High-Byte INDEX ARY
15075  7cd8  9d 01 01        sta stack+1,x
15076  7cdb
15077  7cdb  c8              iny
15078  7cdc  84 0d           sty count       Anzahl DIM
15079  7cde
15080  7cde  20 86 03        jsr chrgot      letztes Zeichen erneut nach ac
15081  7ce1  a4 0d           ldy count       Anzahl DIM
15082  7ce3
15083  7ce3  c9 2c           cmp #','
15084  7ce5  f0 ce           beq indlop
15085  7ce7
15086  7ce7  20 56 79        jsr chkcls      Test auf folg. »)«, sonst Fehler
15087  7cea
15088  7cea  68              pla             valtyp
15089  7ceb  85 0f           sta valtyp      Flag Variablentyp
15090  7ced
15091  7ced  68              pla             dimflg or intflg
15092  7cee  85 10           sta intflg      Flag für Gleitkomma/Integer
15093  7cf0
15094  7cf0  29 7f           and #%01111111
15095  7cf2  85 0e           sta dimflg      Flag DIM/Zeiger suchen
15096  7cf4
15097  7cf4  ===> ARRAY Tabelle nach VAR durchsuchen <===
15098  7cf4
15099  7cf4  a6 31           ldx arytab      ADDRL Zeiger Anfang Feld-Tabelle
15100  7cf6  a5 32           lda arytab+1    ADDRH Zeiger Anfang Feld-Tabelle
15101  7cf8
15102  7cf8
15103  7cf8  ===> Test Obergr. ARY TAB erreicht <===
15104  7cf8
15105  7cf8  86 61     lopfda stx lowtr       ADDRL Zeiger ARY TAB
15106  7cfa  85 62            sta lowtr+1     ADDRH Zeiger ARY TAB
15107  7cfc
15108  7cfc  c5 34           cmp strend+1    ADDRH+1 Obergr. ARRAY Tabelle
15109  7cfe  d0 04           bne lopfdv      VAR NAM in ARY TAB suchen
15110  7d00
15111  7d00  e4 33           cpx strend      ADDRL+1 Obergr. ARRAY Tabelle
15112  7d02  f0 42           beq notfdd      VARNAM n. gefunden, Feld einrichten
15113  7d04
15114  7d04
15115  7d04  ===> VAR NAM in ARY TAB suchen <===
15116  7d04
15117  7d04  a0 00     lopfdv ldy #0          Zeiger auf 1. Zeichen VAR NAM
15118  7d06  20 00 43         jsr indlo1      LDA (lowtr),y aus VAR-RAM
15119  7d09  c8              iny             Zeiger auf 2. Zeichen VAR NAM
15120  7d0a  c5 47           cmp varnam      erstes Zeichen akt. Variablenname
15121  7d0c  d0 07           bne nmary1
15122  7d0e
15123  7d0e  20 00 43        jsr indlo1      LDA (lowtr),y aus VAR-RAM
15124  7d11  c5 48           cmp varnam+1    zweites Zeichen akt. Variablenname
15125  7d13  f0 18           beq gotary      VAR in ARY TAB gefunden
15126  7d15
```

```
15127  7d15  c8          nmary1 iny              Zeiger auf Low-Byte ARY LEN
15128  7d16  20 00 43          jsr indlo1        LDA (lowtr),y aus VAR-RAM
15129  7d19  18                clc
15130  7d1a  65 61              adc lowtr         ADDRL Zeiger in ARY TAB
15131  7d1c  aa                 tax               ADDRL Zeiger auf nächstes ARY
15132  7d1d
15133  7d1d  c8                 iny               Zeiger auf High-Byte ARY LEN
15134  7d1e  20 00 43           jsr indlo1        LDA (lowtr),y aus VAR-RAM
15135  7d21  65 62              adc lowtr+1       ADDRH Zeiger in ARY TAB
15136  7d23  90 d3              bcc lopfda        Test Obergr. ARY TAB erreicht
15137  7d25
15138  7d25
15139  7d25  ===> Behandlung »?bad subscript error« <===
15140  7d25
15141  7d25  a2 12       bserr  ldx #_erbs
15142  7d27  2c                 .byte _skip2
15143  7d28
15144  7d28
15145  7d28  ===> Behandlung »?illegal quantity error« <===
15146  7d28
15147  7d28  a2 0e       fcerr  ldx #_erfc
15148  7d2a
15149  7d2a  ===> ind. Behandlung Fehlermeldung <===
15150  7d2a
15151  7d2a  4c 3c 4d    errgo3 jmp error          ind. Behandlung Fehlermeldung
15152  7d2d
15153  7d2d
15154  7d2d  ===> VAR in ARY TAB gefunden <===
15155  7d2d
15156  7d2d  a2 13       gotary ldx #_erdd
15157  7d2f
15158  7d2f  a5 0e              lda dimflg        Flag DIM/Zeiger suchen
15159  7d31
15160  7d31   <> 0  Feld dim. (1. Zeich. VAR NAM)
15161  7d31    = 0  Zeiger auf Var. suchen
15162  7d31
15163  7d31  d0 f7              bne errgo3        Behandlung »?redim'd array error«
15164  7d33
15165  7d33  20 71 7e           jsr fmaptr        Zeiger 1. Element nach ARYPNT
15166  7d36
15167  7d36  a0 04              ldy #4            Zeiger auf Anzahl DIM in ARY Kopf
15168  7d38  20 00 43           jsr indlo1        LDA (lowtr),y aus VAR-RAM
15169  7d3b  85 79              sta syntmp        Anzahl DIM aus ARY Kopf
15170  7d3d
15171  7d3d  a5 0d              lda count         Anzahl DIM
15172  7d3f  c5 79              cmp syntmp        Anzahl DIM aus ARY Kopf
15173  7d41  d0 e2              bne bserr         Behandlung »?bad subscript error«
15174  7d43
15175  7d43  4c d2 7d           jmp getdef        VARPNT auf gesuchtes Element
15176  7d46
15177  7d46
15178  7d46  ===> VARNAM n. gefunden, Feld einrichten <===
15179  7d46
15180  7d46  20 71 7e   notfdd jsr fmaptr        Zeiger 1. Element nach ARYPNT
15181  7d49  20 17 50           jsr reason        Test Platz in VAR-RAM, sonst
                                                 garb.coll.
15182  7d4c
15183  7d4c  a0 00              ldy #0            Zeiger in ARY Kopf
```

```
15184  7d4e  84 73           sty fbufpt+1     High-Byte Element Länge
15185  7d50
15186  7d50  a2 05           ldx #5           Element Länge
15187  7d52
15188  7d52  a5 47           lda varnam       erstes Zeichen akt. Variablenname
15189  7d54  8d 04 ff        sta lcrd         ROM / VAR-RAM einschalten
15190  7d57  91 61           sta (lowtr),y    ADDRL Zeiger ARY Kopf
15191  7d59  10 01           bpl notflt
15192  7d5b
15193  7d5b  ca              dex              Element Länge
15194  7d5c
15195  7d5c  c8        notflt iny             Zeiger in ARY Kopf
15196  7d5d  a5 48           lda varnam+1     zweites Zeichen akt. Variablenname
15197  7d5f  91 61           sta (lowtr),y    ADDRL Zeiger ARY Kopf
15198  7d61  10 02           bpl stomlt
15199  7d63
15200  7d63  ca              dex
15201  7d64  ca              dex              Element Länge
15202  7d65
15203  7d65  86 72     stomlt stx fbufpt      Element Länge (GK=5 Str=3 Int=2)
15204  7d67
15205  7d67  a5 0d           lda count        Anzahl DIM
15206  7d69  c8              iny
15207  7d6a  c8              iny
15208  7d6b  c8              iny
15209  7d6c  91 61           sta (lowtr),y    in Header bringen
15210  7d6e
15211  7d6e  a2 0b     loppta ldx #11         Low-Byte Vorgabe Anz. Elem.
15212  7d70  a9 00           lda #0           High-Byte Vorgabe Anz. Elem.
15213  7d72
15214  7d72  24 0e           bit dimflg       Flag DIM/Zeiger suchen
15215  7d74
15216  7d74   01xxxxxx  Aufruf durch DIM (1. Zeich. VAR NAM)
15217  7d74   00000000  Aufruf durch ptrget (Zeiger suchen)
15218  7d74
15219  7d74  50 08           bvc notdim
15220  7d76
15221  7d76  ===> Aufruf durch DIM <===
15222  7d76
15223  7d76  68              pla              Low-Byte Anzahl Elemente
15224  7d77  18              clc
15225  7d78  69 01           adc #1
15226  7d7a  aa              tax              Low-Byte Anzahl Elemente
15227  7d7b
15228  7d7b  68              pla              High-Byte Anzahl Elemente
15229  7d7c  69 00           adc #0           High-Byte Anzahl Elemente
15230  7d7e
15231  7d7e  c8        notdim iny
15232  7d7f  91 61           sta (lowtr),y    ADDRL Zeiger ARY Kopf
15233  7d81
15234  7d81  c8              iny              Zeiger auf Low-Byte (high/low!)
15235  7d82  8a              txa              Low-Byte Anzahl Elemente
15236  7d83  91 61           sta (lowtr),y    ADDRL Zeiger ARY Kopf
15237  7d85
15238  7d85  20 3e 7e        jsr umult        DIM Wert mal FBUFPT nach xr/yr=ac
15239  7d88  86 72           stx fbufpt       Low-Byte ARY Länge
15240  7d8a  85 73           sta fbufpt+1     High-Byte ARY Länge
15241  7d8c
```

```
15242  7d8c  a4 24            ldy index1       yr zurück
15243  7d8e
15244  7d8e  c6 0d            dec count        Anzahl DIM
15245  7d90  d0 dc            bne loppta
15246  7d92
15247  7d92  65 5b            adc arypnt+1     ADDRH 1. Element ARY
15248  7d94  b0 67            bcs omerr4       Behandlung »?out of memory error«
15249  7d96
15250  7d96  85 5b            sta arypnt+1     ADDRH Byte hinter ARY
15251  7d98  a8               tay              ADDRH Byte hinter ARY
15252  7d99
15253  7d99  8a               txa              Low-Byte ARY Länge
15254  7d9a  65 5a            adc arypnt       ADDRL 1. Element ARY
15255  7d9c  90 03            bcc grease
15256  7d9e  c8               iny              ADDRH Byte hinter ARY
15257  7d9f  f0 5c            beq omerr4       Behandlung »?out of memory error«
15258  7da1
15259  7da1  20 17 50   grease jsr reason      Test Platz in VAR-RAM, sonst
                                                garb.coll.
15260  7da4
15261  7da4  85 33            sta strend       ADDRL+1 Obergr. ARRAY Tabelle
15262  7da6  84 34            sty strend+1     ADDRH+1 Obergr. ARRAY Tabelle
15263  7da8
15264  7da8  a9 00            lda #0           Füll Byte
15265  7daa  e6 73            inc fbufpt+1     High-Byte ARY Länge
15266  7dac  a4 72            ldy fbufpt       Low-Byte ARY Länge
15267  7dae  f0 05            beq deccur
15268  7db0
15269  7db0
15270  7db0  ===> ARY mit Nullen füllen <===
15271  7db0
15272  7db0  88         zerita dey
15273  7db1  91 5a             sta (arypnt),y
15274  7db3  d0 fb             bne zerita      ARY mit Nullen füllen
15275  7db5
15276  7db5  c6 5b      deccur dec arypnt+1    ADDRH Zeiger in ARY
15277  7db7  c6 73             dec fbufpt+1    High-Byte ARY Länge
15278  7db9  d0 f5             bne zerita      ARY mit Nullen füllen
15279  7dbb
15280  7dbb  ===> ARY Länge nach ARY Kopf <===
15281  7dbb
15282  7dbb  e6 5b             inc arypnt+1    ADDRH Zeiger in ARY
15283  7dbd
15284  7dbd  38                sec
15285  7dbe  a5 33             lda strend      ADDRL+1 Obergr. ARRAY Tabelle
15286  7dc0  e5 61             sbc lowtr       ADDRL Zeiger ARY Kopf
15287  7dc2
15288  7dc2  a0 02             ldy #2          Zeiger Low-Byte Feld Länge
15289  7dc4  91 61             sta (lowtr),y   ADDRL Zeiger ARY Kopf
15290  7dc6
15291  7dc6  a5 34             lda strend+1    ADDRH+1 Obergr. ARRAY Tabelle
15292  7dc8  c8                iny             Zeiger High-Byte Feld Länge
15293  7dc9  e5 62             sbc lowtr+1     ADDRH Zeiger ARY Kopf
15294  7dcb  91 61             sta (lowtr),y   ADDRL Zeiger ARY Kopf
15295  7dcd
15296  7dcd  a5 0e             lda dimflg      Flag DIM/Zeiger suchen
15297  7dcf
15298  7dcf   01xxxxxx  Aufruf durch DIM (1. Zeich. VAR NAM)
```

```
15299  7dcf   00000000  Aufruf durch ptrget (Zeiger suchen)
15300  7dcf
15301  7dcf  d0 6c             bne dimrts
15302  7dd1
15303  7dd1
15303  7dd1  ===> Aufruf durch ptrget mit autom. DIM <===
15304  7dd1
15305  7dd1  c8                iny             Zeiger auf Anzahl DIM in ARY Kopf
15306  7dd2
15307  7dd2
15308  7dd2  ===> VARPNT auf gesuchtes Element <===
15309  7dd2
15310  7dd2  20 00 43   getdef jsr indlo1      Anzahl DIM aus ARY Kopf
15311  7dd5  85 0d             sta count       Anzahl DIM
15312  7dd7
15313  7dd7  a9 00             lda #0
15314  7dd9  85 72             sta fbufpt      Low-Byte Stelle ges. Element
15315  7ddb
15316  7ddb
15317  7ddb  ===> Stelle ges. Elem. in ARY berechnen <===
15318  7ddb
15319  7ddb  85 73      inlpnm sta fbufpt+1    High-Byte Stelle ges. Element
15320  7ddd
15321  7ddd  c8                iny             Zeiger auf High-Byte max. Element
                                               Zahl
15322  7dde
15323  7dde  68                pla             Low-Byte Nummer ges. Element
15324  7ddf  aa                tax             Low-Byte Nummer ges. Element
15325  7de0  85 66             sta facmo       Low-Byte Nummer ges. Element
15326  7de2
15327  7de2  20 00 43          jsr indlo1      LDA (lowtr),y aus VAR-RAM
15328  7de5  85 79             sta syntmp      High-Byte max. Element Zahl
15329  7de7
15330  7de7  68                pla             High-Byte Nummer ges. Element
15331  7de8  85 67             sta faclo       High-Byte Nummer ges. Element
15332  7dea
15333  7dea  c5 79             cmp syntmp      High-Byte max. Element Zahl
15334  7dec  90 12             bcc inlpn2
15335  7dee  d0 0a             bne bserr7      Behandlung »?bad subscript error«
15336  7df0
15337  7df0  c8                iny             Zeiger auf Low-Byte max. Element Zahl
15338  7df1  20 00 43          jsr indlo1      LDA (lowtr),y aus VAR-RAM
15339  7df4
15340  7df4  85 79             sta syntmp      Low-Byte max. Element Zahl
15341  7df6  e4 79             cpx syntmp      mit Low-Byte Nummer ges. Element
15342  7df8  90 07             bcc inlpn1
15343  7dfa
15344  7dfa  ===> Behandlung »?bad subscript error« <===
15345  7dfa
15346  7dfa  4c 25 7d   bserr7 jmp bserr       Behandlung »?bad subscript error«
15347  7dfd
15348  7dfd
15349  7dfd  ===> Behandlung »?out of memory error« <===
15350  7dfd
15351  7dfd  4c 3a 4d   omerr4 jmp omerr       Behandlung »?out of memory error«
15352  7e00
15353  7e00
15354  7e00  c8         inlpn2 iny             Zeiger auf Low-Byte max. Element Zahl
```

```
15355  7e01
15356  7e01  a5 73      inlpn1 lda fbufpt+1     High-Byte Stelle ges. Element
15357  7e03  05 72             ora fbufpt      Low-Byte Stelle ges. Element
15358  7e05  18                clc
15359  7e06  f0 0a             beq addind
15360  7e08
15361  7e08  20 3e 7e          jsr umult       DIM Wert mal FBUFPT nach xr/yr=ac
15362  7e0b
15363  7e0b  8a                txa             Low-Byte Produkt
15364  7e0c  65 66             adc facmo       Low-Byte Nummer ges. Element
15365  7e0e  aa                tax
15366  7e0f
15367  7e0f  98                tya             High-Byte Produkt
15368  7e10  a4 24             ldy index1      Zeiger in DIM Werten zurück
15369  7e12  65 67      addind adc faclo       High-Byte Nummer ges. Element
15370  7e14  86 72             stx fbufpt      Low-Byte Stelle ges. Element
15371  7e16
15372  7e16  c6 0d             dec count       Anzahl DIM
15373  7e18  d0 c1             bne inlpnm      Stelle ges. Elem. in ARY berechnen
15374  7e1a
15375  7e1a  85 73             sta fbufpt+1    High-Byte Stelle ges. Element
15376  7e1c
15377  7e1c  ===> VARPNT auf ges. Element berechnen <===
15378  7e1c
15379  7e1c  a2 05             ldx #5          Low-Byte Element Länge
15380  7e1e
15381  7e1e  a5 47             lda varnam      erstes Zeichen akt. Variablenname
15382  7e20  10 01             bpl notfl1
15383  7e22
15384  7e22  ca                dex
15385  7e23
15386  7e23  a5 48      notfl1 lda varnam+1    zweites Zeichen akt. Variablenname
15387  7e25  10 02             bpl notfl2
15388  7e27
15389  7e27  ca                dex
15390  7e28  ca                dex
15391  7e29
15392  7e29  86 2a      notfl2 stx resmo        Low-Byte Elem. Länge (GK=5 Str=3
                                               Int=2)
15393  7e2b
15394  7e2b  a9 00             lda #0          High-Byte Elem. Länge
15395  7e2d  20 49 7e          jsr umultd      RESMO mal FBUFPT nach xr/yr=ac
15396  7e30
15397  7e30  8a                txa             Low-Byte Offset Elem. in ARY
15398  7e31  65 5a             adc arypnt      ADDRL 1. Element in ARY
15399  7e33  85 49             sta varpnt      ADDRL ges. Element
15400  7e35
15401  7e35  98                tya             High-Byte Offset Elem. in ARY
15402  7e36  65 5b             adc arypnt+1    ADDRH 1. Element in ARY
15403  7e38  85 4a             sta varpnt+1    ADDRH ges. Element
15404  7e3a
15405  7e3a  a8                tay             ADDRH ges. Element
15406  7e3b  a5 49             lda varpnt      ADDRL ges. Element
15407  7e3d
15408  7e3d  60         dimrts rts
15409  7e3e
15410  7e3e
```

```
15411  7e3e  ===> DIM Wert mal FBUFPT nach xr/yr=ac <===
15412  7e3e
15413  7e3e  84 24      umult  sty index1       Puffr yr
15414  7e40
15415  7e40  20 00 43          jsr indlo1       LDA (lowtr),y aus VAR-RAM
15416  7e43  85 2a              sta resmo        Low-Byte max. Element Zahl
15417  7e45
15418  7e45  88                dey              Zeiger auf High-Byte
15419  7e46  20 00 43          jsr indlo1       LDA (lowtr),y aus VAR-RAM
15420  7e49
15421  7e49
15422  7e49  ===> RESMO mal FBUFPT nach xr/yr=ac <===
15423  7e49
15424  7e49  85 2b      umultd sta reslo        RES: M1
15425  7e4b
15426  7e4b  a9 10             lda #16          Zähler für 16 Bit
15427  7e4d  85 5f             sta deccnt
15428  7e4f
15429  7e4f  a2 00             ldx #0           Low-Byte Produkt
15430  7e51  a0 00             ldy #0           High-Byte Produkt
15431  7e53
15432  7e53  8a        umultc  txa
15433  7e54  0a                asl a
15434  7e55  aa                tax              Low-Byte Produkt
15435  7e56
15436  7e56  98                tya
15437  7e57  2a                rol a
15438  7e58  a8                tay              High-Byte Produkt
15439  7e59
15440  7e59  b0 a2             bcs omerr4       Behandlung »?out of memory error«
15441  7e5b
15442  7e5b  ===> nächstes Bit aus Faktor nach CARRY <===
15443  7e5b
15444  7e5b  06 72             asl fbufpt       Low-Byte Faktor
15445  7e5d  26 73             rol fbufpt+1     High-Byte Faktor
15446  7e5f  90 0b             bcc umlcnt       Bit = 0
15447  7e61
15448  7e61  ===> Bit = 1, Addieren <===
15449  7e61
15450  7e61  18                clc
15451  7e62  8a                txa              Low-Byte Produkt
15452  7e63  65 2a             adc resmo        Low-Byte max. Element Zahl
15453  7e65  aa                tax              Low-Byte Produkt
15454  7e66
15455  7e66  98                tya              High-Byte Produkt
15456  7e67  65 2b             adc reslo        High-Byte max. Element Zahl
15457  7e69  a8                tay              High-Byte Produkt
15458  7e6a
15459  7e6a  b0 91             bcs omerr4       Behandlung »?out of memory error«
15460  7e6c
15461  7e6c  c6 5f      umlcnt dec deccnt        Zähler 16 Bit
15462  7e6e  d0 e3             bne umultc
15463  7e70
15464  7e70  60                rts
15465  7e71
15466  7e71
```

```
15467  7e71  ===> Zeiger 1. Element nach ARYPNT <===
15468  7e71
15469  7e71  a5 0d      fmaptr lda count        Anzahl DIM
15470  7e73  0a               asl a            mal zwei
15471  7e74  69 05            adc #_aryhd      Länge ARY Kopf
15472  7e76  65 61            adc lowtr        ADDRL Zeiger ARY TAB
15473  7e78
15474  7e78  a4 62            ldy lowtr+1      ADDRH Zeiger ARY TAB
15475  7e7a  90 01            bcc fmapt1
15476  7e7c  c8               iny
15477  7e7d
15478  7e7d  85 5a      fmapt1 sta arypnt       ADDRL Zeiger in ARY
15479  7e7f  84 5b            sty arypnt+1     ADDRH Zeiger in ARY
15480  7e81
15481  7e81  60               rts
15482  7e82
15483  7e82                   * = $7ffe
15484  7ffe
15485  7ffe  00 4c            .word $4c00
15486  8000
15487  8000
15488  8000                   .end
15489  8000                   .lib basic.functs
15490  8000  put"@basic.functs"
```

```
15492  8000
15493  8000
15494  8000  ===> BASIC-Funktion FRE ($b8) <===
15495  8000
15496  8000  20 f7 87   fre     jsr conint       8 Bit INT CONST aus FAC nach xr
15497  8003  e0 01              cpx #_varbk      VAR RAM
15498  8005
15499  8005  90 05              bcc frefpg       FRE(PGM RAM)
15500  8007  f0 31              beq frefst       FRE(VAR RAM)
15501  8009
15502  8009  ===> Behandlung »?illegal quantity error« <===
15503  8009
15504  8009  4c 28 7d           jmp fcerr        Behandlung »?illegal quantity error«
15505  800c
15506  800c
15507  800c  ===> FRE(PGM RAM) <===
15508  800c
15509  800c  38          frefpg sec
15510  800d  ad 12 12           lda mxmem0       ADDRL Obergr. PGM RAM
15511  8010  ed 10 12           sbc txttop       ADDRL Obergrenze akt. PGM
15512  8013  a8                 tay              Low-Byte Differenz in yr
15513  8014  ad 13 12           lda mxmem0+1     ADDRH Obergr. PGM RAM
15514  8017  ed 11 12           sbc txttop+1     High-Byte Differenz in ac
15515  801a
15516  801a  b0 2b              bcs fref10       (fast) immer
15517  801c
15518  801c  ===> Entschlüsselung einer Mitteilung ($ae63) der C 128 Väter <===
15519  801c
15520  801c   Die Mitteilung kann durch (z.B.) folgende Eingaben im Direkt-
15521  801c   modus auf den Bildschirm gebracht werden:
15522  801c
15523  801c   poke 4624, 255: poke 4625, 255: poke 4626, 68
15524  801c   poke 4627, 59 : poke   53,  44: print fre(0)    <return>
15525  801c
15526  801c  a6 35              ldx fretop       ADDRL Zeiger Anfang String-Variable
15527  801e  e8                inx
15528  801f  c8                iny
15529  8020  85 70              sta arisgn       Vorzeichen Ergebnis Operation
15530  8022
15531  8022  98                tya
15532  8023  38                sec
15533  8024  e9 05              sbc #5
15534  8026  85 71              sta facov        Zähler
15535  8028
15536  8028  a5 71       dcplop lda facov        Zähler
15537  802a  5d 37 ae           eor kydmsg-44,x  xor mit verschlüsseltem Text
15538  802d  45 70              eor arisgn       xor mit Konstante
15539  802f  f0 44              beq val2
15540  8031
15541  8031  20 d2 ff           jsr kbsout       ind. Ausg. ac auf akt. Kanal
                                                 (Sprungv.)
15542  8034  e6 71              inc facov        Zähler
15543  8036  e8                inx              Zeiger in Text
15544  8037  d0 ef              bne dcplop
15545  8039
15546  8039  60                rts
15547  803a
15548  803a
```

```
15549 803a
15549 803a  ===> FRE(VAR RAM) <===
15550 803a
15551 803a  20 ea 92   frefst jsr garba2       Garbage Collection
15552 803d  38                sec
15553 803e  a5 35             lda fretop       ADDRL Zeiger Anfang String-Variable
15554 8040  e5 33             sbc strend       ADDRL+1 Obergr. ARRAY Tabelle
15555 8042  a8                tay              Low-Byte Differenz in yr
15556 8043  a5 36             lda fretop+1     ADDRH Zeiger Anfang String-Variable
15557 8045  e5 34             sbc strend+1     High-Byte Differenz in ac
15558 8047
15559 8047  4c c9 84   fref10 jmp nosflt       16 Bit INT yr/ac nach GK in FAC
15560 804a
15561 804a
15562 804a  ===> BASIC-Funktion VAL ($c5) <===
15563 804a
15564 804a  20 6e 86   val    jsr len1         Stringparam. aus Stapel, Stringlänge
                                               ac
15565 804d  d0 03             bne val1         String Länge >0
15566 804f
15567 804f  ===> VAL Leer String = 0 <===
15568 804f
15569 804f  4c d6 88          jmp zerofc       FAC auf 00 setzen
15570 8052
15571 8052
15572 8052  ===> String auswerten <===
15573 8052
15574 8052  18         val1   clc              String Länge
15575 8053  65 24             adc index1       ADDRL String
15576 8055  85 72             sta fbufpt       ADDRL Zeichen hinter String
15577 8057
15578 8057  a5 25             lda index1+1     ADDRH String
15579 8059  69 00             adc #0
15580 805b  85 73             sta fbufpt+1     ADDRH Zeichen hinter String
15581 805d
15582 805d  a0 00             ldy #>fbufpt
15583 805f  a9 72             lda #<fbufpt
15584 8061  20 ab 03          jsr iram1        LDA (ac),y aus VAR-RAM
15585 8064  48                pha              Zeichen hinter String
15586 8065
15587 8065  98                tya              eom
15588 8066  91 72             sta (fbufpt),y   eom hinter String
15589 8068
15590 8068  ===> Wert String nach FAC <===
15591 8068
15592 8068  20 03 8e          jsr fnchg2
15593 806b
15594 806b  a2 01             ldx #_varbk      VAR RAM
15595 806d  20 22 8d          jsr fin          Zahlenstr. (Bnk.=xr) nach FAC
15596 8070
15597 8070  ===> altes Zeichen hinter String wiederherstellen <===
15598 8070
15599 8070  68                pla              Zeichen hinter String
15600 8071  a0 00             ldy #0           Zeiger
15601 8073  91 72             sta (fbufpt),y   Zeichen hinter String
15602 8075
15603 8075  60         val2   rts
15604 8076
```

```
15605  8076
15606  8076
15606  8076  ===> BASIC-Funktion DEC ($d1) <===
15607  8076
15608  8076  20 6e 86   dcml   jsr len1          Stringparam. aus Stapel, Stringlänge
                                                ac
15609  8079  85 26             sta index2        String Länge
15610  807b
15611  807b  a0 00             ldy #0            Zeiger
15612  807d  84 27             sty index2+1      Stellen Zähler
15613  807f
15614  807f  84 72             sty fbufpt        Low-Byte Ergebnis
15615  8081  84 73             sty fbufpt+1      High-Byte Ergebnis
15616  8083
15617  8083
15618  8083  ===> nächstes Byte aus String <===
15619  8083
15620  8083  c4 26      dec010 cpy index2        String Länge
15621  8085  f0 34             beq dec050        Ergebnis nach GK in FAC
15622  8087
15623  8087  20 b7 03          jsr ind1_1        LDA (index1),y aus VAR-RAM
15624  808a  c8                iny
15625  808b
15626  808b  c9 20             cmp #_spc
15627  808d  f0 f4             beq dec010        nächstes Byte aus String
15628  808f
15629  808f  e6 27             inc index2+1      Stellen Zähler
15630  8091  a6 27             ldx index2+1      Stellen Zähler
15631  8093  e0 05             cpx #5
15632  8095  f0 2b             beq decbad        Behandlung »?illegal quantity error«
15633  8097
15634  8097  c9 30             cmp #'0'
15635  8099  90 27             bcc decbad        Behandlung »?illegal quantity error«
15636  809b
15637  809b  c9 3a             cmp #$3a          ASCII(9) +1
15638  809d  90 0a             bcc dec030        Low-Nibble nach high, ins Ergebnis
15639  809f
15640  809f  c9 41             cmp #'a'
15641  80a1  90 1f             bcc decbad        Behandlung »?illegal quantity error«
15642  80a3
15643  80a3  c9 47             cmp #'g'          ASCII(f) +1
15644  80a5  b0 1b             bcs decbad        Behandlung »?illegal quantity error«
15645  80a7
15646  80a7  ===> Korrektur a-f <===
15647  80a7
15648  80a7  e9 07             sbc #7
15649  80a9
15650  80a9
15651  80a9  ===> Low-Nibble nach high, ins Ergebnis <===
15652  80a9
15653  80a9  e9 2f      dec030 sbc #$2f          ASCII(0) -1
15654  80ab
15655  80ab  0a                asl a
15656  80ac  0a                asl a
15657  80ad  0a                asl a
15658  80ae  0a                asl a
15659  80af
15660  80af  a2 04             ldx #4            Bit Zähler Nibble
```

```
15661  80b1
15662  80b1  0a          dec040 asl a
15663  80b2  26 72              rol fbufpt       Low-Byte Ergebnis
15664  80b4  26 73              rol fbufpt+1     High-Byte Ergebnis
15665  80b6
15666  80b6  ca                  dex              Bit Zähler
15667  80b7  d0 f8               bne dec040
15668  80b9  f0 c8               beq dec010       immer
15669  80bb
15670  80bb
15671  80bb  ===> Ergebnis nach GK in FAC <===
15672  80bb
15673  80bb  a4 72       dec050  ldy fbufpt       Low-Byte Ergebnis
15674  80bd  a5 73               lda fbufpt+1     High-Byte Ergebnis
15675  80bf  4c c9 84            jmp nosflt       16 Bit INT yr/ac nach GK in FAC
15676  80c2
15677  80c2
15678  80c2  ===> Behandlung »?illegal quantity error« <===
15679  80c2
15680  80c2  4c 28 7d    decbad  jmp fcerr        Behandlung »?illegal quantity error«
15681  80c5
15682  80c5
15683  80c5  ===> BASIC-Funktion PEEK ($c2) <===
15684  80c5
15685  80c5  a5 17       peek    lda linnum+1     linnum retten
15686  80c7  48                  pha
15687  80c8
15688  80c8  a5 16               lda linnum
15689  80ca  48                  pha
15690  80cb
15691  80cb  20 da 77            jsr chknum       Test num. VAR, sonst type mism.
15692  80ce  20 15 88            jsr getadr       pos. FAC als 16 Bit INT linnum,
                                                  yr/ac
15693  80d1
15694  80d1  ae d5 03            ldx curbnk       akt. Bank
15695  80d4  a0 00               ldy #>linnum
15696  80d6  a9 16               lda #<linnum
15697  80d8  20 74 ff            jsr kifetv       LDA (FETVEC),y von Bank in xr
                                                  (Sprungv.)
15698  80db
15699  80db  a8                  tay              PEEK Byte
15700  80dc
15701  80dc  68                  pla              alte linnum zurück
15702  80dd  85 16               sta linnum
15703  80df
15704  80df  68                  pla
15705  80e0  85 17               sta linnum+1
15706  80e2
15707  80e2  4c d4 84            jmp sngflt       8 Bit INT yr nach GK in FAC
15708  80e5
15709  80e5
15710  80e5  ===> BASIC-Statement POKE ($97) <===
15711  80e5
15712  80e5  20 03 88    poke    jsr getnum       16 Bit INT nach linnum, Kommatest, 8
                                                  Bit INT nach xr
15713  80e8
15714  80e8  8a                  txa              POKE Byte
15715  80e9  a0 00               ldy #>linnum
```

```
15716  80eb  a2 16           ldx #<linnum
15717  80ed  8e b9 02        stx stashx       STAVEC
15718  80f0  ae d5 03        ldx curbnk       akt. Bank
15719  80f3  4c 77 ff        jmp kistav       STA (STAVEC),y in Bank in xr
                                              (Sprungv.)
15720  80f6
15721  80f6
15722  80f6  ===> BASIC-Funktion ERR$ ($d3) <===
15723  80f6
15724  80f6  20 f7 87  errd   jsr conint       8 Bit INT CONST aus FAC nach xr
15725  80f9  ca               dex              Fehler Nummer
15726  80fa  8a               txa
15727  80fb  c9 29            cmp #41
15728  80fd  b0 37            bcs errerr       Behandlung »?illegal quantity error«
15729  80ff
15730  80ff  20 82 4a         jsr erstup       index2 auf Fehlertext (Nummer in xr)
                                              einstellen
15731  8102
15732  8102  a0 ff            ldy #$ff         Zeiger in Fehler Text
15733  8104  a2 00            ldx #0           Text Länge
15734  8106
15735  8106
15736  8106  ===> Text Zeichen zählen <===
15737  8106
15738  8106  e8         errd1  inx             Text Länge
15739  8107
15740  8107
15741  8107  ===> nächstes Text Zeichen testen <===
15742  8107
15743  8107  c8         errd2  iny             Zeiger
15744  8108  b1 26            lda (index2),y
15745  810a  30 06            bmi errd3        Arb.string für Fehler Text anlegen
15746  810c
15747  810c  c9 20            cmp #_spc
15748  810e  90 f7            bcc errd2        keine Steuerzeichen
15749  8110  b0 f4            bcs errd1        immer
15750  8112
15751  8112
15752  8112  ===> Arb.string für Fehler Text anlegen <===
15753  8112
15754  8112  8a         errd3  txa             Länge Text
15755  8113  20 90 86         jsr strspa       Arb.string Länge ac anlegen
15756  8116
15757  8116  a2 00            ldx #0           Zeiger in String
15758  8118  a0 ff            ldy #$ff         Zeiger in Fehler Text
15759  811a  8d 04 ff         sta lcrd         ROM / VAR-RAM einschalten
15760  811d
15761  811d
15762  811d  ===> Fehler Text nach Arb.string <===
15763  811d
15764  811d  c8         errd4  iny             Zeiger Fehler Text
15765  811e  b1 26            lda (index2),y   Zeichen Fehler Text
15766  8120  c9 20            cmp #_spc
15767  8122  90 f9            bcc errd4        keine Steuerzeichen
15768  8124
15769  8124  20 39 81         jsr swapxy       Inhalt xr / yr tauschen
15770  8127  48               pha              Zeichen Fehler Text
15771  8128
```

```
15772  8128  29 7f           and #%01111111
15773  812a  91 64           sta (facho),y    Arbeitsstring
15774  812c
15775  812c  20 39 81        jsr swapxy       Inhalt xr / yr tauschen
15776  812f  e8              inx              Zeiger String
15777  8130
15778  8130  68              pla              Zeichen Fehler Text
15779  8131  10 ea           bpl errd4        nicht letztes Zeichen, weiter
15780  8133
15781  8133  4c d1 85        jmp chrd1        Descr. Arb.string in Tabelle
15782  8136
15783  8136
15784  8136  ===> Behandlung »?illegal quantity error« <===
15785  8136
15786  8136  4c 28 7d  errerr jmp fcerr       Behandlung »?illegal quantity error«
15787  8139
15788  8139
15789  8139  ===> Inhalt xr / yr tauschen <===
15790  8139
15791  8139  48        swapxy pha
15792  813a
15793  813a  8a              txa
15794  813b  48              pha
15795  813c
15796  813c  98              tya
15797  813d  aa              tax
15798  813e
15799  813e  68              pla
15800  813f  a8              tay
15801  8140
15802  8140  68              pla
15803  8141  60              rts
15804  8142
15805  8142
15806  8142  ===> BASIC-Funktion HEX$ ($d2) <===
15807  8142
15808  8142  20 da 77  hexd   jsr chknum      Test num. VAR, sonst type mism.
15809  8145
15810  8145  a5 16           lda linnum       linnum retten
15811  8147  48              pha
15812  8148
15813  8148  a5 17           lda linnum+1
15814  814a  48              pha
15815  814b
15816  814b  20 15 88        jsr getadr       pos. FAC als 16 Bit INT linnum,
                                              yr/ac
15817  814e
15818  814e  a9 04           lda #4           String Länge
15819  8150  20 90 86        jsr strspa       Arb.string Länge ac anlegen
15820  8153
15821  8153  a0 00           ldy #0           Zeiger in Arbeitsstring
15822  8155  a5 17           lda linnum+1     High-Byte
15823  8157  8d 04 ff        sta lcrd         ROM / VAR-RAM einschalten
15824  815a  20 6b 81        jsr hexit        ac nach HEX in Arb.string
15825  815d
15826  815d  a5 16           lda linnum       Low-Byte
15827  815f  20 6b 81        jsr hexit        ac nach HEX in Arb.string
15828  8162
```

```
15829  8162  68              pla             alte linnum zurück
15830  8163  85 17           sta linnum+1
15831  8165
15832  8165  68              pla
15833  8166  85 16           sta linnum
15834  8168
15835  8168  4c d1 85        jmp chrd1       Descr. Arb.string in Tabelle
15836  816b
15837  816b
15838  816b  ===> ac nach HEX in Arb.string <===
15839  816b
15840  816b  48        hexit pha             High-/Low-Nibble
15841  816c
15842  816c  4a              lsr a           High-Nibble nach low
15843  816d  4a              lsr a
15844  816e  4a              lsr a
15845  816f  4a              lsr a
15846  8170
15847  8170  20 74 81        jsr dohex
15848  8173
15849  8173  68              pla             High-/Low-Nibble
15850  8174
15851  8174  29 0f     dohex and #%00001111
15852  8176  c9 0a           cmp #10
15853  8178  90 02           bcc dohex1
15854  817a
15855  817a  69 06           adc #6          Korrektur a-f
15856  817c  69 30     dohex1 adc #'0'       HEX nach ASCII
15857  817e  91 64           sta (facho),y   Arbeitsstring
15858  8180
15859  8180  c8              iny             Zeiger Arb. String
15860  8181  60              rts
15861  8182
15862  8182
15863  8182  ===> BASIC-Funktion RGR ($cc) <===
15864  8182
15865  8182  20 da 77  rgr    jsr chknum      Test num. VAR, sonst type mism.
15866  8185  20 8c 81        jsr unpgpm      Graphik Modus 40/80 Zeichen BS
                                             berechnen
15867  8188
15868  8188  a8              tay             GRAPHIC Modus (0-9)
15869  8189  4c d4 84        jmp sngflt      8 Bit INT yr nach GK in FAC
15870  818c
15871  818c
15872  818c  ===> Graphik Modus 40/80 Zeichen BS berechnen <===
15873  818c
15874  818c  a5 d8      unpgpm lda graphm     Flag Text/Graphik/Split-BS
15875  818e
15876  818e    00000000  Text                                  GRAPH 0
15877  818e    00100000  HI-RES                                GRAPH 1
15878  818e    01100000  geteilter BS HI-RES/Text               GRAPH 2
15879  818e    10100000  HI-RES Multi-Color                    GRAPH 3
15880  818e    11100000  geteilter BS HI-RES Multi-Color/Text  GRAPH 4
15881  818e
15882  818e  ===> Bit5 (HI RES) nach CARRY <===
15883  818e
15884  818e  18              clc
15885  818f  2a              rol a
```

```
15886  8190  2a              rol a
15887  8191  2a              rol a
15888  8192
15889  8192  69 00           adc #0
15890  8194
15891  8194
15892  8194
15892  8194  ===> Werte 40 Zeichen BS <===
15893  8194
15894  8194    0  Text                                  GRAPH 0
15895  8194    1  HI-RES                                GRAPH 1
15896  8194    2  geteilter BS HI-RES/Text              GRAPH 2
15897  8194    3  HI-RES Multi-Color                    GRAPH 3
15898  8194    4  geteilter BS HI-RES Multi-Color/Text  GRAPH 4
15899  8194
15900  8194  24 d7           bit mode        BS-Modus 40/80-Zeichen
15901  8196  10 02           bpl unpgp1      40 Zeichen BS
15902  8198
15903  8198  ===> 80 Zeichen BS <===
15904  8198
15905  8198  69 05           adc #5
15906  819a
15907  819a  ===> Werte 80 Zeichen BS <===
15908  819a
15909  819a    5  Text                                  GRAPH 0
15910  819a    6  HI-RES                                GRAPH 1
15911  819a    7  geteilter BS HI-RES/Text              GRAPH 2
15912  819a    8  HI-RES Multi-Color                    GRAPH 3
15913  819a    9  geteilter BS HI-RES Multi-Color/Text  GRAPH 4
15914  819a
15915  819a
15916  819a  60        unpgp1 rts
15917  819b
15918  819b
15919  819b  ===> BASIC-Funktion RCLR ($cd) <===
15920  819b
15921  819b  20 f7 87   rclr   jsr conint      8 Bit INT CONST aus FAC nach xr
15922  819e
15923  819e   0 = Hintergrund (40 Zeichen BS)
15924  819e   1 = Vordergrund
15925  819e   2 = Zusatzfarbe 1
15926  819e   3 = Zusatzfarbe 2
15927  819e   4 = Rahmenfarbe
15928  819e   5 = Text Farbe (80 Zeichen BS)
15929  819e   6 = Hintergrund (80 Zeichen BS)
15930  819e
15931  819e  20 45 a8          jsr mapio       I/O-Chips einblenden
15932  81a1  ca                dex             Farbquelle
15933  81a2  30 15             bmi rclr10      0 = Hintergrund Farbe (40 Zeichen
                                                BS)
15934  81a4
15935  81a4  ca                dex             Farbquelle
15936  81a5  30 1a             bmi rclr20      1 = Vordergrund Farbe
15937  81a7
15938  81a7  ca                dex             Farbquelle
15939  81a8  30 1c             bmi rclr30      2 = Zusatzfarbe 1
15940  81aa
15941  81aa  ca                dex             Farbquelle
```

```
15942  81ab  30 1e            bmi rclr40       3 = Zusatzfarbe 2
15943  81ad
15944  81ad  ca               dex              Farbquelle
15945  81ae  30 20            bmi rclr50       4 = Rahmen Farbe (40 Zeichen BS)
15946  81b0
15947  81b0  ca               dex              Farbquelle
15948  81b1  30 23            bmi rclr60       5 = Text Farbe
15949  81b3
15950  81b3  ca               dex              Farbquelle
15951  81b4  30 28            bmi rclr70       6 = Hintergrund Farbe (89 Zeichen BS)
15952  81b6
15953  81b6  ===> Behandlung »?illegal quantity error« <===
15954  81b6
15955  81b6  4c 28 7d          jmp fcerr        Behandlung »?illegal quantity error«
15956  81b9
15957  81b9
15958  81b9  ===> 0 = Hintergrund Farbe (40 Zeichen BS) <===
15959  81b9
15960  81b9  ad 21 d0   rclr10 lda vicbkg       VIC 8564 Hintergrundfarbe
15961  81bc
15962  81bc   7654....  unbenutzt
15963  81bc   ....3210  Hintergrund Farbe (0-15)
15964  81bc
15965  81bc  29 7f            and #%01111111
15966  81be  4c ec 81          jmp rclr90
15967  81c1
15968  81c1
15969  81c1  ===> 1 = Vordergrund Farbe <===
15970  81c1
15971  81c1  a5 86      rclr20 lda forgnd       Farbcode Vordergrund
15972  81c3  4c ec 81          jmp rclr90
15973  81c6
15974  81c6
15975  81c6  ===> 2 = Zusatzfarbe 1 <===
15976  81c6
15977  81c6  a5 84      rclr30 lda mulco1       Multi-Color Farbe 1
15978  81c8  4c ec 81          jmp rclr90
15979  81cb
15980  81cb
15981  81cb  ===> 3 = Zusatzfarbe 2 <===
15982  81cb
15983  81cb  a5 85      rclr40 lda mulco2       Zusatzfarbe 2
15984  81cd  4c ec 81          jmp rclr90
15985  81d0
15986  81d0
15987  81d0  ===> 4 = Rahmen Farbe (40 Zeichen BS) <===
15988  81d0
15989  81d0  ad 20 d0   rclr50 lda vicbrd       VIC 8564 Rahmenfarbe
15990  81d3
15991  81d3   7654....  unbenutzt15992  81d3   ....3210  Rahmenfarbe (0-15)
15993  81d3
15994  81d3  4c ec 81          jmp rclr90
15995  81d6
15996  81d6
15997  81d6  ===> 5 = Text Farbe <===
15998  81d6
15999  81d6  a5 f1      rclr60 lda attrib       akt. Attribut
16000  81d8
```

```
16001  81d8   7654....  Attribute
16002  81d8   ....3210  RGB und Helligkeit
16003  81d8
16004  81d8  24 d7            bit mode        BS-Modus 40/80-Zeichen
16005  81da  10 10            bpl rclr90      40 Zeichen BS
16006  81dc  30 08            bmi rclr80      80 Zeichen BS
16007  81de
16008  81de
16009  81de  ===> 6 = Hintergrund Farbe (89 Zeichen BS) <===
16010  81de
16011  81de  a9 1a      rclr70 lda #26         Vordergr. / Hintergr. Farbe
16012  81e0  8d 00 d6         sta vdcadr      VDC 8563 Adreß-Register
16013  81e3
16014  81e3  ad 01 d6         lda vdcdat      VDC 8563 Daten-Register
16015  81e6
16016  81e6   7654....  Vordergrund Farbe, wenn Bit6 von Reg. 25 =0
16017  81e6   ....3210  Hintergrund Farbe Zeichen/Rahmen
16018  81e6
16019  81e6  29 0f      rclr80 and #%00001111  Farbcode
16020  81e8  aa               tax
16021  81e9
16022  81e9  bd f3 81         lda unadju,x    8563 unadjust
16023  81ec  29 0f      rclr90 and #%00001111
16024  81ee
16025  81ee  a8               tay             Farbnummer -1
16026  81ef  c8               iny             Farbnummer
16027  81f0  4c d4 84         jmp sngflt      8 Bit INT yr nach GK in FAC
16028  81f3
16029  81f3
16030  81f3  ===> 8563 unadjust <===
16031  81f3
16032  81f3  00         unadju .byte 0         schwarz
16033  81f4  0c                .byte 12        grau
16034  81f5  06                .byte 6         blau
16035  81f6  0e                .byte 14        hellblau
16036  81f7  05                .byte 5         dunkelgrün
16037  81f8  0d                .byte 13        hellgrün
16038  81f9  0b                .byte 11        dunkelgrau
16039  81fa  03                .byte 3         grün
16040  81fb  02                .byte 2         rot
16041  81fc  0a                .byte 10        hellrot
16042  81fd  08                .byte 8         hellbraun
16043  81fe  04                .byte 4         violett
16044  81ff  09                .byte 9         braun
16045  8200  07                .byte 7         gelb
16046  8201  0f                .byte 15        hellgrau
16047  8202  01                .byte 1         weiß
16048  8203
16049  8203
16050  8203  ===> BASIC-Funktion JOY ($cf) <===
16051  8203
16052  8203  20 f7 87   joy    jsr conint      8 Bit INT CONST aus FAC nach xr
16053  8206  ca               dex             Nummer Joystick
16054  8207  e0 02            cpx #2
16055  8209  b0 34            bcs joy30
16056  820b
16057  820b  8a               txa             Nummer Joystick
16058  820c  49 01            eor #1
```

```
16059  820e  aa              tax
16060  820f  08              php
16061  8210
16062  8210  20 45 a8        jsr mapio       I/O-Chips einblenden
16063  8213
16064  8213  78              sei
16065  8214  ad 00 dc        lda d1pra       6526 #1
16066  8217
16067  8217   76543210  Spalten Tastatur Matrix (OUT)
16068  8217
16069  8217  48              pha
16070  8218
16071  8218  a0 ff           ldy #%11111111
16072  821a  8c 00 dc        sty d1pra       6526 #1
16073  821d
16074  821d
16075  821d  ===> Port entprellen <===
16076  821d
16077  821d  bd 00 dc   joy10  lda d1pra,x    6526 #1
16078  8220
16079  8220   76......  D1PRA: Auswahl Paddle Set A/B
16080  8220   ...43210  D1PRA: Joystick 0
16081  8220   ...43210  ROWS : Joystick 1
16082  8220
16083  8220  dd 00 dc        cmp d1pra,x     6526 #1
16084  8223  d0 f8           bne joy10       Port entprellen
16085  8225
16086  8225  aa              tax             Joystick Port
16087  8226
16088  8226  68              pla             Tastatur Matrix
16089  8227  8d 00 dc        sta d1pra       6526 #1
16090  822a
16091  822a  8a              txa             Joystick Port
16092  822b  28              plp
16093  822c
16094  822c  29 0f           and #%00001111
16095  822e
16096  822e   ....0101  4 = diag. re. hinten
16097  822e   ....0110  2 = diag. re. vorn
16098  822e   ....0111  3 = rechts
16099  822e   ....1000  0 = Mittelstellung
16100  822e   ....1001  6 = diag. li. hinten
16101  822e   ....1010  8 = diag. li. vorn
16102  822e   ....1011  7 = links
16103  822e   ....1100  0 = Mittelstellung
16104  822e   ....1101  5 = hinten
16105  822e   ....1110  1 = vorn
16106  822e   ....1111  0 = Mittelstellung
16107  822e
16108  822e  a8              tay
16109  822f
16110  822f  b9 3d 82        lda joytab-5,y
16111  8232  a8              tay             Joystick Wert
16112  8233
16113  8233  ===> Abfrage Feuerknopf <===
16114  8233
16115  8233  8a              txa             Joystick Port
16116  8234  29 10           and #%00010000  Feuerknopf
```

```
16117  8236  d0 04            bne joy20
16118  8238
16119  8238  98               tya              Joystick Wert
16120  8239  09 80            ora #%10000000
16121  823b  a8               tay              Joystick Wert
16122  823c
16123  823c  4c d4 84   joy20  jmp sngflt       8 Bit INT yr nach GK in FAC
16124  823f
16125  823f
16126  823f  ===> Behandlung »?illegal quantity error« <===
16127  823f
16128  823f  4c 28 7d   joy30  jmp fcerr        Behandlung »?illegal quantity error«
16129  8242
16130  8242
16131  8242  04         joytab .byte 4
16132  8243  02                .byte 2
16133  8244  03                .byte 3
16134  8245  00                .byte 0
16135  8246  06                .byte 6
16136  8247  08                .byte 8
16137  8248  07                .byte 7
16138  8249  00                .byte 0
16139  824a  05                .byte 5
16140  824b  01                .byte 1
16141  824c  00                .byte 0
16142  824d
16143  824d
16144  824d  ===> BASIC-Funktion POT ($ce $02) <===
16145  824d
16146  824d  20 56 79   pot    jsr chkcls       Test auf folg. »)«, sonst Fehler
16147  8250
16148  8250  ===> Nummer Game Paddle einlesen <===
16149  8250
16150  8250  20 f7 87          jsr conint       8 Bit INT CONST aus FAC nach xr
16151  8253  ca                dex
16152  8254  e0 04             cpx #4
16153  8256  b0 53            bcs poterr
16154  8258
16155  8258  20 45 a8          jsr mapio        I/O-Chips einblenden
16156  825b
16157  825b  a0 00            ldy #0           Vorgabe Paddle 2/4: A/D Wandler 0
16158  825d  8a               txa              Paddle Nummer (1-4)
16159  825e  a2 40            ldx #%01000000   Vorgabe Paddle 1/2
16160  8260
16161  8260  4a               lsr a
16162  8261  90 01           bcc pot10        Paddle 2/4: A/D Wandler 0
16163  8263
16164  8263  ===> Paddle 1/3: A/D Wandler 1 <===
16165  8263
16166  8263  c8               iny              Zeiger auf A/D Wandler
16167  8264
16168  8264  4a         pot10  lsr a
16169  8265  90 02           bcc pot20        Paddle 1/2
16170  8267
16171  8267  ===> Paddle 3/4 <===
16172  8267
16173  8267  a2 80            ldx #%10000000
16174  8269
```

```
16175  8269  8e b1 12   pot20  stx tmpot1       Flag Paddle 1/2 bzw. 3/4
16176  826c  08                php
16177  826d
16178  826d  78                sei
16179  826e  ad 00 dc          lda d1pra        6526 #1
16180  8271  48                pha
16181  8272
16182  8272  8e 00 dc          stx d1pra        6526 #1
16183  8275
16184  8275   7.......  1 = Paddle 3/4
16185  8275   .6......  1 = Paddle 1/2
16186  8275
16187  8275
16188  8275  ===> Warteschleife 1.3 ms <===
16189  8275
16190  8275  a2 00             ldx #0
16191  8277  e8          pot30  inx
16192  8278  d0 fd             bne pot30
16193  827a
16194  827a
16195  827a  ===> Port entprellen <===
16196  827a
16197  827a  b9 19 d4   pot40  lda sid+potx,y   SID A/D Wandler 1
16198  827d
16199  827d   76543210  A/D Wandler Wert (0-255)
16200  827d
16201  827d  d9 19 d4          cmp sid+potx,y   SID A/D Wandler 1
16202  8280  d0 f8             bne pot40        Port entprellen
16203  8282
16204  8282  8d b2 12          sta tmpot2       A/D Wandler Wert
16205  8285
16206  8285  a2 00             ldx #0           Zeiger auf
16207  8287
16208  8287  2c b1 12          bit tmpot1       Flag Paddle 1/2 bzw. 3/4
16209  828a  30 01             bmi pot50
16210  828c
16211  828c  e8                inx              Zeiger auf
16212  828d
16213  828d  a9 04       pot50 lda #%00000100   Maske Feuerknopf
16214  828f
16215  828f  88                dey
16216  8290  30 01             bmi pot60        Paddle 1/3
16217  8292
16218  8292  ===> Paddle 2/4 <===
16219  8292
16220  8292  0a                asl a            %00001000
16221  8293
16222  8293
16223  8293  ===> Abfrage Feuerknopf <===
16224  8293
16225  8293  a0 ff       pot60 ldy #%11111111
16226  8295  8c 00 dc          sty d1pra        6526 #1
16227  8298
16228  8298  c8                iny              High-Byte Ergebnis
16229  8299
16230  8299  3d 00 dc          and d1pra,x      6526 #1
16231  829c  d0 01             bne pot70        d1pra wiederherstellen
16232  829e
```

```
16233  829e  c8               iny             High-Byte Ergebnis
16234  829f
16235  829f
16236  829f  ===> d1pra wiederherstellen <===
16237  829f
16238  829f  68         pot70 pla
16239  82a0  8d 00 dc         sta d1pra       6526 #1
16240  82a3
16241  82a3
16241  82a3  ===> Übergabe Wert berechnen <===
16242  82a3
16243  82a3  98               tya             High-Byte (Feuerknopf)
16244  82a4  ac b2 12         ldy tmpot2      Low-Byte (A/D Wert)
16245  82a7  28               plp
16246  82a8  4c c9 84         jmp nosflt      16 Bit INT yr/ac nach GK in FAC
16247  82ab
16248  82ab
16249  82ab  ===> Behandlung »?illegal quantity error« <===
16250  82ab
16251  82ab  4c 28 7d   poterr jmp fcerr      Behandlung »?illegal quantity error«
16252  82ae
16253  82ae
16254  82ae  ===> BASIC-Funktion PEN ($ce $04) <===
16255  82ae
16256  82ae  20 56 79   pen    jsr chkcls      Test auf folg. »)«, sonst Fehler
16257  82b1  20 f7 87         jsr conint      8 Bit INT CONST aus FAC nach xr
16258  82b4
16259  82b4   0 = Abszisse Light Pen (60-320)
16260  82b4   1 = Ordinate Light Pen (50-200)
16261  82b4   2 = Spalte Light Pen (80 Zeichen BS)
16262  82b4   3 = Zeile  Light Pen (80 Zeichen BS)
16263  82b4   4 = Light Pen Aktiv.Flag
16264  82b4
16265  82b4  e0 05            cpx #5
16266  82b6  b0 f3            bcs poterr
16267  82b8
16268  82b8  e0 02            cpx #2
16269  82ba  b0 1a            bcs pen20
16270  82bc
16271  82bc  ===> Koord. Light Pen übergeben <===
16272  82bc
16273  82bc  bc e9 11         ldy lpabsz,x    Kopie Light-Pen, x-Wert (Abszisse)
16274  82bf  8c b1 12         sty tmpot1      Abszisse / Ordinate
16275  82c2
16276  82c2  a9 00            lda #0          High-Byte Übergabe Wert
16277  82c4  9d e9 11         sta lpabsz,x    Kopie rücksetzen
16278  82c7
16279  82c7  e0 00            cpx #0
16280  82c9  d0 05            bne pen10
16281  82cb
16282  82cb  ===> Abszisse verdoppeln <===
16283  82cb
16284  82cb  0e b1 12         asl tmpot1      Flag Paddle 1/2 bzw. 3/4
16285  82ce  69 00            adc #0          High-Byte Übergabe Wert
16286  82d0  ac b1 12   pen10 ldy tmpot1      Low-Byte Übergabe Wert
16287  82d3  4c c9 84         jmp nosflt      16 Bit INT yr/ac nach GK in FAC
16288  82d6
16289  82d6
```

```
16290  82d6  20 45 a8   pen20  jsr mapio        I/O-Chips einblenden
16291  82d9  e0 04             cpx #4
16292  82db  f0 10             beq pen40        Übergabe Akt.Flag Light Pen <===
16293  82dd
16294  82dd  ===> Bildzeile/Bildspalte Light Pen Übergeben <===
16295  82dd
16296  82dd  a0 11             ldy #17          Spalte Light Pen
16297  82df  e0 02             cpx #2
16298  82e1  f0 01             beq pen30
16299  82e3
16300  82e3  88                dey              Zeile Light Pen
16301  82e4
16302  82e4  8c 00 d6   pen30  sty vdcadr       VDC 8563 Adreß-Register
16303  82e7
16304  82e7  ac 01 d6          ldy vdcdat       VDC 8563 Daten-Register
16305  82ea
16306  82ea  16:  76543210  vert. Light Pen Pos. (in Bildzeilen +1)
16307  82ea  17:  76543210  hor.  Light Pen Pos. (in Bildspalten +1)
16308  82ea
16309  82ea  4c d4 84          jmp sngflt       8 Bit INT yr nach GK in FAC
16310  82ed
16311  82ed
16312  82ed  ===> Übergabe Akt.Flag Light Pen <=== <===
16313  82ed
16314  82ed  ad 00 d6   pen40  lda vdcadr       VDC 8563 Adreß-Register
16315  82f0  a0 00             ldy #0           Übergabe Wert 0
16316  82f2  29 40             and #%01000000
16317  82f4
16318  82f4   x0xxxxxx  nach Lesen Reg. 16/17 Light Pen
16319  82f4   x1xxxxxx  Light Pen Strobe erkannt
16320  82f4
16321  82f4  f0 01             beq pen50        Akt.Flag nicht gesetzt
16322  82f6
16323  82f6  c8                iny              Übergabe Wert 1
16324  82f7
16325  82f7  4c d4 84   pen50  jmp sngflt       8 Bit INT yr nach GK in FAC
16326  82fa
16327  82fa
16328  82fa  ===> BASIC-Funktion POINTER ($ce $0a) <===
16329  82fa
16330  82fa  20 80 03   pointr jsr chrget       nächstes Zeichen nach ac
16331  82fd  20 59 79          jsr chkopn       Test auf folg. »(«, sonst Fehler
16332  8300
16333  8300  20 3c 7b          jsr isletc       Test ac auf ALPHA (CARRY=1)
16334  8303  90 16             bcc ponter       Behandlung »?syntax error«
16335  8305
16336  8305  20 af 7a          jsr ptrget       folg. Variable suchen/anlegen
16337  8308  aa                tax              ADDRL Variable
16338  8309
16339  8309  98                tya
16340  830a  48                pha              ADDRH Variable
16341  830b
16342  830b  20 56 79          jsr chkcls       Test auf folg. »)«, sonst Fehler
16343  830e
16344  830e  8a                txa
16345  830f  a8                tay              ADDRL Variable
16346  8310
16347  8310  68                pla              ADDRH Variable
```

```
16348  8311
16349  8311  ===> Test auf System Variable <===
16350  8311
16351  8311  c9 03            cmp #>zero       Konstante Null
16352  8313  d0 03            bne point1
16353  8315
16354  8315  ===> Übergabe Wert System Variable: 0 <===
16355  8315
16356  8315  a9 00            lda #0           High-Byte Übergabe Wert
16357  8317  a8               tay              Low-Byte Übergabe Wert
16358  8318
16359  8318  4c c9 84   point1 jmp nosflt       16 Bit INT yr/ac nach GK in FAC
16360  831b
16361  831b
16362  831b  ===> Behandlung »?syntax error« <===
16363  831b
16364  831b  4c 6c 79   ponter jmp snerr        Behandlung »?syntax error«
16365  831e
16366  831e
16367  831e  ===> BASIC-Funktion RSPRITE ($ce $06) <===
16368  831e
16369  831e  20 f7 87   rsprit jsr conint       8 Bit INT CONST aus FAC nach xr
16370  8321  ca               dex              Sprite Nummer -1
16371  8322  e0 10            cpx #16
16372  8324  b0 32            bcs rspr20
16373  8326
16374  8326  8a               txa
16375  8327  48               pha              Sprite Nummer -1
16376  8328
16377  8328  ===> Modus RSPRITE einlesen <===
16378  8328
16379  8328  20 5c 79         jsr chkcom       Test auf folg. »Komma« sonst Fehler
16380  832b  20 f4 87         jsr getbyt       8 Bit INT üb. frmnum nach xr
16381  832e  20 56 79         jsr chkcls       Test auf folg. »)«, sonst Fehler
16382  8331
16383  8331   0 = Flag Sprite aktiviert (1=akt.)
16384  8331   1 = Sprite Farbe          (1-16)
16385  8331   2 = Priorität Flag        (0=Prior.)
16386  8331   3 = Expand x Flag         (1=Expand)
16387  8331   4 = Expand y Flag         (1=Expand)
16388  8331   5 = MULTICOLOR FLAG       (1=MUL.COL)
16389  8331
16390  8331  e0 06            cpx #6
16391  8333  b0 23            bcs rspr20
16392  8335
16393  8335  68               pla
16394  8336  a8               tay              Sprite Nummer -1
16395  8337
16396  8337  20 45 a8         jsr mapio        I/O-Chips einblenden
16397  833a  b9 27 d0         lda vicspc,y     VIC 8564 Farbe Sprite0
16398  833d
16399  833d   7654....  unbenutzt
16400  833d   ....3210  Farbe Sprite (0-15)
16401  833d
16402  833d  29 0f            and #%00001111
16403  833f
16404  833f  18               clc
16405  8340  69 01            adc #1           Farbe Sprite (1-16)
```

```
16406  8342
16407  8342  e0 01           cpx #1
16408  8344  f0 0e           beq rspr10       Übergabe Sprite Farbe
16409  8346
16410  8346  bd 5b 83        lda rspmod,x     Tab. VIC 8564 Reg. RSPRITE
16411  8349  aa              tax              VIC 8564 Register
16412  834a  b9 b3 6c        lda sbits,y      Tabelle Maskenbits Bit0-7
16413  834d  3d 00 d0        and vic,x        VIC 8564 Basisadresse
16414  8350  f0 02           beq rspr10       übergabe Wert 0
16415  8352
16416  8352  a9 01           lda #1           übergabe Wert 1
16417  8354
16418  8354  a8        rspr10 tay
16419  8355  4c d4 84        jmp sngflt       8 Bit INT yr nach GK in FAC
16420  8358
16421  8358
16422  8358  ===> Behandlung »?illegal quantity error« <===
16423  8358
16424  8358  4c 28 7d  rspr20 jmp fcerr        Behandlung »?illegal quantity error«
16425  835b
16426  835b
16427  835b  ===> Tab. VIC 8564 Reg. RSPRITE <===
16428  835b
16429  835b  15        rspmod .byte 21         Sprite Enable Register
16430  835c  27               .byte 39         Farbe Sprite (unbenutzt)
16431  835d  1b               .byte 27         Priorität Sprite7-0
16432  835e  1d               .byte 29         Expand x Sprite7-0
16433  835f  17               .byte 23         Expand y Sprite7-0
16434  8360  1c               .byte 28         MULTICOLOR Sprite7-0
16435  8361
16436  8361
16437  8361  ===> BASIC-Funktion RSPCOLOR ($ce $07) <===
16438  8361
16439  8361  20 56 79   rspcol jsr chkcls      Test auf folg. »)«, sonst Fehler
16440  8364  20 f7 87         jsr conint       8 Bit INT CONST aus FAC nach xr
16441  8367  ca               dex              Zusatzfarbe
16442  8368  e0 02            cpx #2
16443  836a  b0 0d            bcs rsperr       Behandlung »?illegal quantity error«
16444  836c
16445  836c  20 45 a8         jsr mapio        I/O-Chips einblenden
16446  836f  bd 25 d0         lda vicmc0,x     Sprite-Multicolor-Register
16447  8372
16448  8372   7654....  unbenutzt
16449  8372   ....3210  Zusatzfarbe 1/2 (0-15)
16450  8372
16451  8372  29 0f            and #%00001111
16452  8374  a8               tay              Farb Nummer (0-15)
16453  8375
16454  8375  c8               iny              Farb Nummer (1-16)
16455  8376  4c d4 84         jmp sngflt       8 Bit INT yr nach GK in FAC
16456  8379
16457  8379
16458  8379  ===> Behandlung »?illegal quantity error« <===
16459  8379
16460  8379  4c 28 7d   rsperr jmp fcerr        Behandlung »?illegal quantity error«
16461  837c
16462  837c
```

```
16463  837c  ===> BASIC-Funktion BUMP ($ce $03) <===
16464  837c
16465  837c  20 56 79   bump   jsr chkcls       Test auf folg. »)«, sonst Fehler
16466  837f  20 f7 87          jsr conint       8 Bit INT CONST aus FAC nach xr
16467  8382  ca                dex              Modus -1
16468  8383  e0 02             cpx #2
16469  8385  b0 0d             bcs bmperr       Behandlung »?illegal quantity error«
16470  8387
16471  8387  78                sei
16472  8388  bc e7 11          ldy sscoll,x     Flags Sprite Kollision mit Sprite7-0
16473  838b
16474  838b
16474  838b  ===> Flag rücksetzen <===
16475  838b
16476  838b  a9 00             lda #0
16477  838d  9d e7 11          sta sscoll,x     Flags Sprite Kollision mit Sprite7-0
16478  8390
16479  8390  58                cli
16480  8391  4c d4 84          jmp sngflt       8 Bit INT yr nach GK in FAC
16481  8394
16482  8394
16483  8394  ===> Behandlung »?illegal quantity error« <===
16484  8394
16485  8394  4c 28 7d   bmperr jmp fcerr        Behandlung »?illegal quantity error«
16486  8397
16487  8397
16488  8397  ===> BASIC-Funktion RSPPOS ($ce $05) <===
16489  8397
16490  8397  20 f7 87   rsppos jsr conint       8 Bit INT CONST aus FAC nach xr
16491  839a  ca                dex              Sprite Nummer -1
16492  839b  e0 10             cpx #16
16493  839d  b0 3f             bcs rspper       Behandlung »?illegal quantity error«
16494  839f
16495  839f  8a                txa
16496  83a0  48                pha              Sprite Nummer -1
16497  83a1
16498  83a1  ===> Modus RSPPOS einlesen <===
16499  83a1
16500  83a1  20 5c 79          jsr chkcom       Test auf folg. »Komma« sonst Fehler
16501  83a4  20 f4 87          jsr getbyt       8 Bit INT üb. frmnum nach xr
16502  83a7  20 56 79          jsr chkcls       Test auf folg. »)«, sonst Fehler
16503  83aa
16504  83aa   0 = Übergabe akt. Abszisse
16505  83aa   1 = Übergabe akt. Ordinate
16506  83aa   2 = Übergabe akt. Geschwindigkeit
16507  83aa
16508  83aa  e0 03             cpx #3
16509  83ac  b0 30             bcs rspper       Behandlung »?illegal quantity error«
16510  83ae
16511  83ae  68                pla
16512  83af  a8                tay              Sprite Nummer -1
16513  83b0
16514  83b0  e0 02             cpx #2
16515  83b2  d0 09             bne rspps1       Übergabe akt. Sprite Koord.
16516  83b4
```

```
16519  83b4  be d9 6d          ldx sproff,y     Tabelle Offsets für Parameter
                                                Sprite0-7
16520  83b7  bc 7e 11          ldy sspeed,x     Konstante Schritte / BASIC-IRQ
16521  83ba  4c d4 84          jmp sngflt       8 Bit INT yr nach GK in FAC
16522  83bd
16523  83bd
16524  83bd  ===> Übergabe akt. Sprite Koord. <===
16525  83bd
16526  83bd  78          rspps1 sei
16527  83be  b9 b3 6c          lda sbits,y      Tabelle Maskenbits Bit0-7
16528  83c1  2d e6 11          and spmsb        MSB-Position Sprite7-0
16529  83c4  f0 02             beq rspps2       MSB Abszisse =0
16530  83c6
16531  83c6  a9 01             lda #1           MSB Abszisse =1
16532  83c8  48          rspps2 pha              MSB Abszisse Sprite7-0
16533  83c9
16534  83c9  98                tya              Sprite Nummer -1
16535  83ca  0a                asl a
16536  83cb  a8                tay              Zeiger Abszisse in Pos.tab.
16537  83cc
16538  83cc  8a                txa              Modus RSPPOS
16539  83cd  4a                lsr a
16540  83ce  90 05             bcc rspps3       Übergabe akt. Abszisse Sprite
16541  83d0
16542  83d0  ===> Übergabe akt. Ordinate Sprite <===
16543  83d0
16544  83d0  c8                iny              Zeiger Ordinate in Pos.Tab.
16545  83d1  68                pla              MSB Abszisse Sprite7-0
16546  83d2
16547  83d2  a9 00             lda #0           MSB Ordinate immer 0
16548  83d4  48                pha              MSB Ordinate Sprite7-0
16549  83d5
16550  83d5
16551  83d5  ===> Übergabe Koord. Sprite <===
16552  83d5
16553  83d5  b9 d6 11    rspps3 lda postab,y     Abszisse Sprite
16554  83d8  58                 cli
16555  83d9
16556  83d9  a8                 tay             Low-Byte Koord.
16557  83da  68                 pla             High-Byte Koord.
16558  83db  4c c9 84           jmp nosflt      16 Bit INT yr/ac nach GK in FAC
16559  83de
16560  83de
16561  83de  ===> Behandlung »?illegal quantity error« <===
16562  83de
16563  83de  4c 28 7d    rspper jmp fcerr       Behandlung »?illegal quantity error«
16564  83e1
16565  83e1
16566  83e1  ===> BASIC-Funktion XOR ($ce $08) <===
16567  83e1
16568  83e1  a5 16       xor    lda linnum      linnum retten
16569  83e3  48                 pha
16570  83e4
16571  83e4  a5 17              lda linnum+1
16572  83e6  48                 pha
16573  83e7
```

```
16574  83e7  ===> 16 Bit INT #1 einlesen <===
16575  83e7
16576  83e7  20 da 77          jsr chknum       Test num. VAR, sonst type mism.
16577  83ea  20 15 88          jsr getadr       pos. FAC als 16 Bit INT linnum, yr/ac
16578  83ed  48                pha              High-Byte 16 Bit INT #1
16579  83ee
16580  83ee  98                tya
16581  83ef  48                pha              Low-Byte 16 Bit INT #1
16582  83f0
16583  83f0  ===> 16 Bit INT #2 einlesen <===
16584  83f0
16585  83f0  20 0f 88          jsr comwrd       Kommatest, 16 Bit INT nach linnum
16586  83f3  20 56 79          jsr chkcls       Test auf folg. »)«, sonst Fehler
16587  83f6
16588  83f6  68                pla              Low-Byte 16 Bit INT #1
16589  83f7  45 16             eor linnum       Low-Byte 16 Bit INT #
16590  83f9  a8                tay              Low-Byte 16 Bit INT #1 XOR #2
16591  83fa
16592  83fa  68                pla              High-Byte 16 Bit INT #1
16593  83fb  45 17             eor linnum+1     High-Byte 16 Bit INT #2
16594  83fd  20 c9 84          jsr nosflt       16 Bit INT yr/ac nach GK in FAC
16595  8400
16596  8400  68                pla              alte linnum zurück
16597  8401  85 17             sta linnum+1
16598  8403
16599  8403  68                pla
16600  8404  85 16             sta linnum
16601  8406
16602  8406  60                rts
16603  8407
16604  8407
16605  8407  ===> BASIC-Funktion RWINDOW ($ce $09) <===
16606  8407
16607  8407  20 56 79   rwndow jsr chkcls       Test auf folg. »)«, sonst Fehler
16608  840a  20 f7 87          jsr conint       8 Bit INT CONST aus FAC nach xr
16609  840d
16610  840d   0 = Übergabe letzte Zeile akt. BS Fenster
16611  840d   1 = Übergabe letzte Spalte akt. BS Fenster
16612  840d   2 = Übergabe 40/80 (Zeichen BS)
16613  840d
16614  840d  e0 02             cpx #2
16615  840f  f0 14             beq rwndo2
16616  8411  b0 1e             bcs rwnder       Behandlung »?illegal quantity error«
16617  8413
16618  8413  e0 00             cpx #0
16619  8415  d0 07             bne rwndo1       Übergabe letzte Spalte akt. BS
                                                Fenster
16620  8417
16621  8417  ===> Übergabe letzte Zeile akt. BS Fenster <===
16622  8417
16623  8417  a5 e4             lda scbot        BS-Fenster: unterste Zeile
16624  8419  38                sec
16625  841a  e5 e5             sbc sctop        BS-Fenster: oberste Zeile
16626  841c  b0 0f             bcs rwndo3       immer
16627  841e
16628  841e
```

```
16629  841e  ===> Übergabe letzte Spalte akt. BS Fenster <===
16630  841e
16631  841e  a5 e7      rwndo1 lda scrt         BS-Fenster: Spalte rechter Rand
16632  8420  38               sec
16633  8421  e5 e6             sbc sclf         BS-Fenster: Spalte linker Rand
16634  8423  b0 08             bcs rwndo3       immer
16635  8425
16636  8425
16637  8425  a9 28      rwndo2 lda #40          Übergabe Wert 40
16638  8427
16639  8427  24 d7             bit mode         BS-Modus 40/80-Zeichen
16640  8429  10 02             bpl rwndo3       40 Zeichen BS
16641  842b
16642  842b  a9 50             lda #80          Übergabe Wert 80
16643  842d
16644  842d  a8         rwndo3 tay              Übergabe Wert
16645  842e  4c d4 84          jmp sngflt       8 Bit INT yr nach GK in FAC
16646  8431
16647  8431
16648  8431  ===> Behandlung »?illegal quantity error« <===
16649  8431
16650  8431  4c 28 7d   rwnder jmp fcerr        Behandlung »?illegal quantity error«
16651  8434
16652  8434
16653  8434                   .end
16654  8434                   .lib basic.math_1
16655  8434  put@basic.math_1
```

```
16657  8434
16658  8434
16659  8434  ===> Speicherformat von Gleitkommazahlen <===
16660  8434
16661  8434   Gleitkommazahlen werden als Dualzahlen mit 32 Bit breiter Mantisse
16662  8434   und 8 Bit breitem Exponenten dargestellt, beanspruchen also 5 Bytes:
16663  8434
16664  8434                          EX M4 M3 M2 M1
16665  8434
16666  8434
16667  8434   EX            8 Bit Exponent
16668  8434   ==
16669  8434                  Die Arithmetik des C128 modifiziert den Exponenten
16670  8434                  zur Basis 2 durch Addition der Konstanten 129 ($81).
16671  8434                  Dies vereinfacht Manipulationen des Exponenten.
16672  8434
16673  8434                  Vereinbarungsgemäß erhält eine Gleitkommazahl den
16674  8434                  Wert 0, wenn der modifizierte  Exponent den Wert Null
16675  8434                  besitzt.
16676  8434
16677  8434
16678  8434   M4 M3 M2 M1    32 Bit-Mantisse
16679  8434   ===========
16680  8434                  LSB (niedrigstwertiges Bit): Bit0 von M1
16681  8434
16682  8434                  MSB (höchstwertiges Bit)   : Bit7 von M4
16683  8434
16684  8434                      Da das MSB einer Dualzahl immer =1 ist, wird das
16685  8434                      MSB zur Darstellung des Vorzeichens benutzt:
16686  8434
16687  8434                  SG  (Vorzeichen)           : Bit7 von M4: 0 = pos.
16688  8434                                                            1 = neg.
16689  8434
16690  8434                      Zum Rechnen muß nach Merken des Vorzeichens
16691  8434                      das MSB auf 1 gesetzt werden.
16692  8434
16693  8434
16694  8434   Folgendes BASIC-Programm wandelt Gleitkommazahlen vom
16695  8434   Speicherformat nach dezimal:
16696  8434
16697  8434
16698  8434    1000 input "EX M4 M3 M2 M1"; f$
16699  8434    1010 if len(f$) <> 10 then 1000
16700  8434    1020 :
16701  8434    1030 ex = dec(mid$(f$,1,2))
16702  8434    1040 m4 = dec(mid$(f$,3,2))
16703  8434    1050 m3 = dec(mid$(f$,5,2))
16704  8434    1060 m2 = dec(mid$(f$,7,2))
16705  8434    1070 m1 = dec(mid$(f$,9,2))
16706  8434    1080 :
16707  8434    1090 x  = -sgn((m4 and 128) -0.5) * 2 ß (ex - 129) *
16708  8434         (1 + ((m4 and 127) + (m3 +à(m2 + m1/256) /256) /256) /128)
16709  8434    1110 print x
16710  8434    1120 :
16711  8434    1130 goto 1000
16712  8434
16713  8434
16714  8434
```

```
16714  8434  ===> Arithmetik mit Gleitkommazahlen <===
16715  8434
16716  8434
16717  8434   Zur Verarbeitung von Gleitkommazahlen verwendet der C128 zwei
16718  8434   6 Byte breite Register (FLOATING POINT ACCU = FAC), in die er
16719  8434   die 5 Bytes einer Gleitkommazahl ablegt:
16720  8434
16721  8434               FAC                ARG
16722  8434
16723  8434     EX       facexp   $63       argexp  $6a
16724  8434     M4       facho    $64       argho   $6b
16725  8434     M3       facmoh   $65       argmoh  $6c
16726  8434     M2       facmo    $66       argmo   $6d
16727  8434     M1       faclo    $67       arglo   $6e
16728  8434     SG       facsgn   $68       argsgn  $6f
16729  8434
16730  8434   In SG wird das Vorzeichen der Mantisse verwaltet (Bit7), während
16731  8434   das MSB in M4 rekonstruiert wird (Bit7 immer =1).
16732  8434
16733  8434   Zusätzlich benutzen die beiden FACs:
16734  8434
16735  8434             arisgn   $70       Vorzeichen Ergebnis
16736  8434             facov    $71       Rundungsbyte
16737  8434
16738  8434   Ergebnisse von Operationen werden immer in FAC abgelegt.
16739  8434
16740  8434
16741  8434   ===> BASIC-Funktion RND ($bb) <===
16742  8434
16743  8434   20 57 8c   rnd    jsr sign          Flag für FAC NULL, PLUS, MINUS nach
                                                 ac
16744  8437
16745  8437    >0
16746  8437    =0  RND aus TIMER Werten berechnen
16747  8437    <0
16748  8437
16749  8437   30 31      rnd0   bmi rnd1          negativ
16750  8439   d0 1a             bne qsetnr        positiv
16751  843b
16752  843b   ===> RND aus TIMER Werten berechnen <===
16753  843b
16754  843b   20 45 a8          jsr mapio         I/O-Chips einblenden
16755  843e
16756  843e   ad 06 dc          lda d1t2l         CIA1 TIMER B low
16757  8441   85 64             sta facho         FAC: M4
16758  8443
16759  8443   ad 07 dc          lda d1t2h         CIA1 TIMER B high
16760  8446   85 66             sta facmo         FAC: M2
16761  8448
16762  8448   ad 04 dc          lda d1t1l         CIA1 TIMER A low
16763  844b   85 65             sta facmoh        FAC: M3
16764  844d
16765  844d   ad 05 dc          lda d1t1h         CIA1 TIMER A high
16766  8450   85 67             sta faclo         FAC: M1
16767  8452
16768  8452   4c 7a 84          jmp strnex
16769  8455
16770  8455
```

```
16771  8455
16771  8455  ===> Zufallszahl (pos. Argument) <===
16772  8455
16773  8455  a9 1b      qsetnr lda #<rndx       GK Variable RND
16774  8457  a0 12             ldy #>rndx       GK Variable RND
16775  8459  20 d4 8b         jsr movfm        RAM Var. ac/yr nach FAC
16776  845c
16777  845c  a9 90             lda #<rmulc      GK-Konstante 11 879 546 für RND
16778  845e  a0 84             ldy #>rmulc      GK-Konstante 11 879 546 für RND
16779  8460  20 08 8a         jsr rommlt       ROM Konst. ac/yr mit FAC mult.
16780  8463
16781  8463  a9 95             lda #<raddc      GK-Konstante 3.927 677 74 e-8 für RND
16782  8465  a0 84             ldy #>raddc      GK-Konstante 3.927 677 74 e-8 für RND
16783  8467  20 12 8a         jsr romadd       ROM Konst. ac/yr zu FAC addieren
16784  846a
16785  846a
16786  846a  ===> Zufallszahl (neg. Argument) <===
16787  846a
16788  846a  a6 67      rnd1   ldx faclo        FAC: M1
16789  846c  a5 64             lda facho        FAC: M4
16790  846e
16791  846e  85 67             sta faclo        FAC: M1
16792  8470  86 64             stx facho        FAC: M4
16793  8472
16794  8472  a6 65             ldx facmoh       FAC: M3
16795  8474  a5 66             lda facmo        FAC: M2
16796  8476
16797  8476  85 65             sta facmoh       FAC: M3
16798  8478  86 66             stx facmo        FAC: M2
16799  847a
16800  847a  a9 00      strnex lda #0
16801  847c  85 68             sta facsgn       FAC: SG
16802  847e
16803  847e  a5 63             lda facexp       FAC: EX
16804  8480  85 71             sta facov        Rundungsbyte
16805  8482
16806  8482  a9 80             lda #_expcn-1
16807  8484  85 63             sta facexp       FAC: EX
16808  8486
16809  8486  20 b6 88         jsr normal       FAC normalisieren
16810  8489
16811  8489  a2 1b             ldx #<rndx       GK Variable RND
16812  848b  a0 12             ldy #>rndx       GK Variable RND
16813  848d  4c 00 8c         jmp movmf        FAC runden, nach RAM xr/yr
16814  8490
16815  8490
16816  8490  ===> GK-Konstante 11 879 546 für RND <===
16817  8490
16818  8490  98 35 44   rmulc  .byte $98, $35, $44, $7a, $00
16818  8493  7a 00
16819  8495
16820  8495
16821  8495  ===> GK-Konstante 3.927 677 74 e-8 für RND <===
16822  8495
16823  8495  68 28 b1   raddc  .byte $68, $28, $b1, $46, $00
16823  8498  46 00
16824  849a
16825  849a
```

```
16826  849a  ===> GK-Konstante -32 768 <===
16827  849a
16828  849a  90 80 00   n32768 .byte $90, $80, $00, $00, $00
16828  849d  00 00
16829  849f
16830  849f
16831  849f  ===> Wandlung GK/INT <===
16832  849f
16833  849f  20 b4 84   flpint jsr ayint        Wandlung FAC nach sgnd. INT
16834  84a2  a5 66             lda facmo        FAC: M2
16835  84a4  a4 67             ldy faclo        FAC: M1
16836  84a6  60                rts
16837  84a7
16838  84a7
16839  84a7  ===> folg. Variable als pos. INT nach FAC <===
16840  84a7
16841  84a7  20 80 03   intidx jsr chrget       nächstes Zeichen nach ac
16842  84aa  20 ef 77          jsr frmevl       folgenden Ausdruck auswerten
16843  84ad
16844  84ad
16845  84ad  ===> Wandlung FAC nach pos. INT <===
16846  84ad
16847  84ad  20 da 77   posint jsr chknum       Test num. VAR, sonst type mism.
16848  84b0  a5 68             lda facsgn       FAC: SG
16849  84b2  30 0d             bmi nonono       negativ
16850  84b4
16851  84b4
16852  84b4  ===> Wandlung FAC nach sgnd. INT <===
16853  84b4
16854  84b4  a5 63      ayint  lda facexp       FAC: EX
16855  84b6
16856  84b6  ===> Test auf Bereich -32768 bis 32767 <===
16857  84b6
16858  84b6  c9 90             cmp #_expcn+15   Exponent von 32 768
16859  84b8  90 0c             bcc qintgo       kleiner
16860  84ba
16861  84ba  a9 9a             lda #<n32768     GK-Konstante -32 768
16862  84bc  a0 84             ldy #>n32768     GK-Konstante -32 768
16863  84be  20 87 8c          jsr fcomp        FAC mit Var. ac/yr vergleichen
16864  84c1  f0 03      nonono beq qintgo
16865  84c3
16866  84c3  4c 28 7d          jmp fcerr        Behandlung »?illegal quantity error«
16867  84c6
16868  84c6
16869  84c6  4c c7 8c   qintgo jmp qint         FAC GK nach INT
16870  84c9
16871  84c9
16872  84c9  ===> 16 Bit INT yr/ac nach GK in FAC <===
16873  84c9
16874  84c9  20 e5 84   nosflt jsr stoint       16 Bit INT yr/ac nach FAC
16875  84cc  38                sec              verhindert Negierung
16876  84cd  4c 75 8c          jmp floatc       FAC normalisieren
16877  84d0
16878  84d0
16879  84d0  ===> BASIC-Funktion POS ($b9) <===
16880  84d0
16881  84d0  38         pos    sec
16882  84d1  20 8d 92          jsr cplot        Low-Byte akt. Spalte BS nach yr
```

```
16883  84d4
16884  84d4
16885  84d4  ===> 8 Bit INT yr nach GK in FAC <===
16886  84d4
16887  84d4  a9 00      sngflt lda #0           High-Byte nach ac
16888  84d6  4c 3c 79          jmp givayv      16 Bit INT yr/ac nach GK in FAC
16889  84d9
16890  84d9
16891  84d9  ===> Test Direktmodus, Behandl. »?illegal direct error« <===
16892  84d9
16893  84d9  24 7f      errdir bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
16894  84db  30 12             bmi storts
16895  84dd
16896  84dd  ===> Behandlung »?illegal direct error« <===
16897  84dd
16898  84dd  a2 15             ldx #_erid
16899  84df  2c                .byte _skip2
16900  84e0
16901  84e0
16902  84e0  ===> Behandlung »?undefined function error« <===
16903  84e0
16904  84e0  a2 1b      errguf ldx #_eruf
16905  84e2  4c 3c 4d          jmp error       ind. Behandlung Fehlermeldung
16906  84e5
16907  84e5
16908  84e5  ===> 16 Bit INT yr/ac nach FAC <===
16909  84e5
16910  84e5  a2 00      stoint ldx #0           numerisch
16911  84e7  86 0f             stx valtyp      Flag Variablentyp
16912  84e9
16913  84e9  85 64             sta facho       FAC: M4
16914  84eb
16915  84eb  84 65             sty facmoh      FAC: M3
16916  84ed
16917  84ed  a2 90             ldx #_expcn+15  Exponent 15
16918  84ef  60        storts rts
16919  84f0
16920  84f0
16921  84f0  ===> Test Direktmodus, sonst Fehler <===
16922  84f0
16923  84f0  24 7f      errind bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
16924  84f2  30 01             bmi dmoerr      Run-Modus, Fehler -> Sprung
16925  84f4
16926  84f4  60                rts
16927  84f5
16928  84f5
16929  84f5  ===> Behandlung »?direct mode only error« <===
16930  84f5
16931  84f5  a2 22      dmoerr ldx #_eroid
16932  84f7  4c 3c 4d          jmp error       ind. Behandlung Fehlermeldung
16933  84fa
16934  84fa
16935  84fa  ===> BASIC-Statement DEF ($96) <===
16936  84fa
16937  84fa  20 28 85   def     jsr getfnm      FN Variable suchen / anlegen
16938  84fd  20 d9 84          jsr errdir      Test Direktmodus, Behandl. »?illegal
                                               direct error«
16939  8500  20 59 79          jsr chkopn      Test auf folg. »(«, sonst Fehler
```

```
16940  8503
16941  8503  a9 80           lda #$8
16942  8505  85 12           sta sublfg      Integer-Variable sperren
16943  8507
16944  8507  20 af 7a        jsr ptrget      folg. Variable suchen/anlegen
16945  850a  20 da 77        jsr chknum      Test num. VAR, sonst type mism.
16946  850d  20 56 79        jsr chkcls      Test auf folg. »)«, sonst Fehler
16947  8510
16948  8510  a9 b2           lda #_eqult     = TOKEN
16949  8512  20 5e 79        jsr synchr      Test auf Zeichen in ac, sonst Fehler
16950  8515
16951  8515  48              pha             Zeichen hinter =
16952  8516
16953  8516  a5 4a           lda varpnt+1    ADDRH Zeiger auf FN Variable
16954  8518  48              pha
16955  8519
16956  8519  a5 49           lda varpnt      ADDRL Zeiger auf FN Variable
16957  851b  48              pha
16958  851c
16959  851c  a5 3e           lda txtptr+1    ADDRH Programm-Zeiger
16960  851e  48              pha
16961  851f
16962  851f  a5 3d           lda txtptr      ADDRL Programm-Zeiger
16963  8521  48              pha
16964  8522
16965  8522  20 8f 52        jsr data        txtptr auf Trennzeichen ;BASIC-
                                             Statement DATA ($83)
16966  8525  4c a0 85        jmp deffin      FN-Zeiger von Stapel
16967  8528
16968  8528
16969  8528  ===> FN Variable suchen / anlegen <===
16970  8528
16971  8528  a9 a5     getfnm lda #_fnt      FN Token
16972  852a  20 5e 79        jsr synchr      Test auf Zeichen in ac, sonst Fehler
16973  852d  09 80           ora #$80
16974  852f  85 12           sta sublfg      Integer-Variable sperren
16975  8531
16976  8531  20 b6 7a        jsr ptrgt2      Variable suchen oder anlegen
16977  8534  85 50           sta defpnt      ADDRL Zeiger FN Variable
16978  8536  84 51           sty defpnt+1    ADDRH Zeiger FN Variable
16979  8538
16980  8538  4c da 77        jmp chknum      Test num. VAR, sonst type mism.
16981  853b
16982  853b
16983  853b  ===> FN auswerten <===
16984  853b
16985  853b  20 28 85   fndoev jsr getfnm    FN Variable suchen / anlegen
16986  853e
16987  853e  a5 51           lda defpnt+1    ADDRH Zeiger FN Variable
16988  8540  48              pha
16989  8541
16990  8541  a5 50           lda defpnt      ADDRL Zeiger FN Variable
16991  8543  48              pha
16992  8544
16993  8544  20 50 79        jsr parchk      Test auf folg.»(' ')«, sonst Fehler
16994  8547  20 da 77        jsr chknum      Test num. VAR, sonst type mism.
16995  854a
16996  854a  68              pla             ADDRL Zeiger FN Variable
```

```
16997  854b  85 50           sta defpnt      ADDRL Zeiger FN Variable
16998  854d
16999  854d  68              pla             ADDRH Zeiger FN Variabl
17000  854e  85 51           sta defpnt+1    ADDRH Zeiger FN Variable
17001  8550
17002  8550  a0 02           ldy #2
17003  8552  20 ce 42        jsr inddef      LDA (defpnt),y aus VAR-RAM
17004  8555  85 49           sta varpnt      ADDRL Zeiger auf Variable im RAM
17005  8557
17006  8557  aa              tax
17007  8558  c8              iny
17008  8559  20 ce 42        jsr inddef      LDA (defpnt),y aus VAR-RAM
17009  855c  f0 82           beq errguf      Behandlung »?undefined function
                                             error«
17010  855e
17011  855e  85 4a           sta varpnt+1    ADDRH Zeiger auf Variable im RAM
17012  8560
17013  8560  c8              iny
17014  8561  a9 49     defstf lda #<varpnt   ADDRL Zeiger auf Variable im RAM
17015  8563  20 ab 03        jsr iram1       LDA (ac),y aus VAR-RAM
17016  8566  48              pha             Wert der FN Variable Auf Stapel legen
17017  8567  88              dey
17018  8568  10 f7           bpl defstf
17019  856a
17020  856a  a4 4a           ldy varpnt+1    ADDRH Zeiger auf Variable im RAM
17021  856c  8d 04 ff        sta lcrd        ROM / VAR-RAM einschalten
17022  856f
17023  856f  20 00 8c        jsr movmf       FAC runden, nach RAM xr/yr
17024  8572
17025  8572  a5 3e           lda txtptr+1    ADDRH chrget-Zeiger
17026  8574  48              pha
17027  8575
17028  8575  a5 3d           lda txtptr      ADDRL chrget-Zeiger
17029  8577  48              pha
17030  8578
17031  8578  20 ce 42        jsr inddef      LDA (defpnt),y aus VAR-RAM
17032  857b  85 3d           sta txtptr      ADDRL chrget-Zeiger
17033  857d
17034  857d  c8              iny
17035  857e  20 ce 42        jsr inddef      LDA (defpnt),y aus VAR-RAM
17036  8581  85 3e           sta txtptr+1    ADDRH chrget-Zeiger
17037  8583
17038  8583  a5 4a           lda varpnt+1    ADDRH Zeiger auf Variable im RAM
17039  8585  48              pha
17040  8586
17041  8586  a5 49           lda varpnt      ADDRL Zeiger auf Variable im RAM
17042  8588  48              pha
17043  8589
17044  8589  20 d7 77        jsr frmnum      folg. num. Variable nach FAC
17045  858c
17046  858c  68              pla
17047  858d  85 50           sta defpnt      ADDRL Zeiger FN Variable
17048  858f
17049  858f  68              pla
17050  8590  85 51           sta defpnt+1    ADDRH Zeiger FN Variable
17051  8592
17052  8592  20 86 03        jsr chrgot      letztes Zeichen erneut nach ac
17053  8595  f0 03           beq defst1      Trennzeichen
```

```
17054  8597
17055  8597
17055  8597  ===> Behandlung »?syntax error« <===
17056  8597
17057  8597  4c 6c 79          jmp snerr        Behandlung »?syntax error«
17058  859a
17059  859a
17060  859a  68         defst1 pla
17061  859b  85 3d             sta txtptr       ADDRL chrget-Zeiger
17062  859d
17063  859d  68               pla
17064  859e  85 3e             sta txtptr+1     ADDRH chrget-Zeiger
17065  85a0
17066  85a0
17067  85a0  ===> FN-Zeiger von Stapel <===
17068  85a0
17069  85a0  a0 00      deffin ldy #0
17070  85a2  8d 04 ff          sta lcrd         ROM / VAR-RAM einschalten
17071  85a5
17072  85a5  68         defflp pla
17073  85a6  91 50             sta (defpnt),y   FN Variable
17074  85a8  c8                iny              wiederherstellen
17075  85a9  c0 05             cpy #5
17076  85ab  d0 f8             bne defflp
17077  85ad
17078  85ad  60                rts
17079  85ae
17080  85ae
17081  85ae                    .end
17082  85ae                    .lib basic.string
17083  85ae  put"@basic.string"
```

```
17085  85ae
17086  85ae
17087  85ae  ===> BASIC-Funktion STR$ ($c4) <===
17088  85ae
17089  85ae  20 da 77   strd   jsr chknum       Test num. VAR, sonst type mism.
17090  85b1
17091  85b1  a0 00             ldy #0
17092  85b3  20 44 8e          jsr foutc        FAC nach String, Adr. nach ac/yr
17093  85b6  68                pla
17094  85b7  68                pla
17095  85b8  a9 ff      timstr lda #<fbuffr-1
17096  85ba  a0 00             ldy #>fbuffr-1
17097  85bc  4c 9a 86          jmp strlit       String (Descr.ac/yr) nach Arb.string
17098  85bf
17099  85bf
17100  85bf  ===> BASIC-Funktion CHR$ ($c7) <===
17101  85bf
17102  85bf  20 f7 87   chrd   jsr conint       8 Bit INT CONST aus FAC nach xr
17103  85c2  8a                txa
17104  85c3  48                pha
17105  85c4
17106  85c4  a9 01             lda #1           Stringlänge
17107  85c6  20 90 86          jsr strspa       Arb.string Länge ac anlegen
17108  85c9
17109  85c9  68                pla              Byte vom Stapel holen
17110  85ca  a0 00             ldy #0
17111  85cc  8d 04 ff          sta lcrd         ROM / VAR-RAM einschalten
17112  85cf
17113  85cf  91 64             sta (dsctmp+1),y und in Stringbereich kopieren
17114  85d1
17115  85d1
17116  85d1  ===> Descr. Arb.string in Tabelle <===
17117  85d1
17118  85d1  68         chrd1  pla              Rücksprungadresse
17119  85d2  68                pla              verwerfen
17120  85d3  4c e3 86          jmp putnew       String-Descriptor in Tabelle
17121  85d6
17122  85d6
17123  85d6  ===> BASIC-Funktion LEFT$ ($c8) <===
17124  85d6
17125  85d6  20 4d 86   leftd  jsr pream        Stringadresse u. Parameter vom
                                                Stapel
17126  85d9  48                pha              LEFT$ Parameter
17127  85da
17128  85da  20 d8 42          jsr inddpt       LDA (dscpnt),y aus VAR-RAM
17129  85dd  85 79             sta syntmp       Stringlänge
17130  85df
17131  85df  68                pla              LEFT$ Parameter
17132  85e0  c5 79             cmp syntmp       Stringlänge
17133  85e2  98                tya              0 = Anfangsspalte
17134  85e3
17135  85e3
17136  85e3  ===> LEFT$ - RIGHT$ <===
17137  85e3
17138  85e3  90 05      rleft  bcc rleft1       kleiner
17139  85e5
17140  85e5
```

```
17140  85e5  ===> Original-Stringlänge übernehmen <===
17141  85e5
17142  85e5  20 d8 42          jsr inddpt       LDA (dscpnt),y aus VAR-RAM
17143  85e8  aa                tax              neue Stringlänge
17144  85e9  98                tya              0 = Anfangsspalte
17145  85ea
17146  85ea  48          rleft1 pha              Anfangsspalte (Quellstring)
17147  85eb
17148  85eb  8a          rleft2 txa              Stringlänge
17149  85ec
17150  85ec  48          rleft3 pha              Stringlänge
17151  85ed  20 90 86          jsr strspa       Arb.string Länge ac anlegen
17152  85f0
17153  85f0  a5 52             lda dscpnt       ADDRL Zeiger String-Descr.
17154  85f2  a4 53             ldy dscpnt+1     ADDRH Zeiger String-Descr.
17155  85f4  20 85 87          jsr fretmp       Textstring übernehmen
17156  85f7
17157  85f7  68                pla              Stringlänge
17158  85f8  a8                tay
17159  85f9
17160  85f9  68                pla              Anfangsspalte
17161  85fa  18                clc
17162  85fb  65 24             adc index1       +ADDRL Quellstring
17163  85fd  85 24             sta index1
17164  85ff
17165  85ff  90 02             bcc pulmor
17166  8601
17167  8601  e6 25             inc index1+1
17168  8603
17169  8603  98          pulmor tya              Stringlänge
17170  8604  20 63 87          jsr movdo        String index1 in Arbeitsstring
                                                kopieren
17171  8607  4c e3 86          jmp putnew       String-Descriptor in Tabelle
17172  860a
17173  860a
17174  860a  ===> BASIC-Funktion RIGHT$ ($c9) <===
17175  860a
17176  860a  20 4d 86    rightd jsr pream       Stringadresse u. Parameter vom
                                                Stapel
17177  860d  48                pha              RIGHT$ Parameter
17178  860e
17179  860e  20 d8 42          jsr inddpt       LDA (dscpnt),y aus VAR-RAM
17180  8611  85 79             sta syntmp       Stringlänge
17181  8613
17182  8613  68                pla              RIGHT$ Parameter
17183  8614  18                clc              - 1
17184  8615  e5 79             sbc syntmp       - Stringlänge
17185  8617  49 ff             eor #$ff         invertiert = Anfangsspalte
17186  8619  4c e3 85          jmp rleft        LEFT$ - RIGHT$
17187  861c
17188  861c
17189  861c  ===> BASIC-Funktion MID$ ($ca) <===
17190  861c
17191  861c  a9 ff       midd   lda #$ff
17192  861e  85 67             sta indice+1
17193  8620
17194  8620  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
17195  8623  c9 29             cmp #')'
```

```
17196  8625  f0 06          beq mid2        nur 1 MID$ Parameter
17197  8627
17198  8627  20 5c 79       jsr chkcom      Test auf folg. »Komma« sonst Fehler
17199  862a  20 f4 87       jsr getbyt      1. MID$ Parameter nach xr
17200  862d
17201  862d  20 4d 86  mid2  jsr pream       Stringadresse u. Parameter vom
                                            Stapel
17202  8630  f0 53          beq gofuc       Behandlung »?illegal quantity error«
17203  8632
17204  8632  ca              dex             Spaltennummer -1
17205  8633  8a              txa             Zeiger auf 1. Spalte
17206  8634  48              pha             in Quellstring
17207  8635
17208  8635  a2 00           ldx #$00
17209  8637  48              pha             nochmal 1. Spalte
17210  8638
17211  8638  20 d8 42        jsr inddpt      LDA (dscpnt),y aus VAR-RAM
17212  863b  85 79           sta syntmp      Stringlänge
17213  863d
17214  863d  68              pla             1. Spalte in Quellstring
17215  863e  18              clc
17216  863f  e5 79           sbc syntmp      - Stringlänge
17217  8641  b0 a8          bcs rleft2      nicht negativ
17218  8643
17219  8643  49 ff          eor #$ff        neue Stringlänge berechnen
17220  8645  c5 67          cmp indice+1
17221  8647  90 a3          bcc rleft3      < 255
17222  8649
17223  8649  a5 67          lda indice+1    = 255
17224  864b  b0 9f          bcs rleft3      immer
17225  864d
17226  864d
17227  864d  ===> Stringadresse u. Parameter vom Stapel <===
17228  864d
17229  864d  20 56 79  pream  jsr chkcls     Test auf folg. »)«, sonst Fehler
17230  8650
17231  8650  68              pla
17232  8651  a8              tay             ADDRH Rücksprung
17233  8652
17234  8652  68              pla
17235  8653  85 57          sta jmper+1     ADDRL Rücksprung
17236  8655
17237  8655  68              pla             Rücksprungadresse Funktions-Routine
17238  8656  68              pla             verwerfen
17239  8657
17240  8657  68              pla
17241  8658  aa              tax             Parameter
17242  8659
17243  8659  68              pla
17244  865a  85 52          sta dscpnt      ADDRL Zeiger String-Descr.
17245  865c
17246  865c  68              pla
17247  865d  85 53          sta dscpnt+1    ADDRH Zeiger String-Descr.
17248  865f
17249  865f  a5 57          lda jmper+1     ADDRL Rücksprung
17250  8661  48              pha
17251  8662
17252  8662  98              tya             ADDRH Rücksprung
```

```
17253  8663  48              pha
17254  8664
17255  8664  a0 00           ldy #0
17256  8666
17257  8666  8a              txa             1. Parameter
17258  8667  60              rts
17259  8668
17260  8668
17261  8668  ===> BASIC-Funktion LEN ($c3) <===
17262  8668
17263  8668  20 6e 86  len    jsr len1        Stringparam. aus Stapel, Stringlänge
                                             ac
17264  866b  4c d4 84         jmp sngflt      8 Bit INT yr nach GK in FAC
17265  866e
17266  866e
17267  866e  ===> Stringparam. aus Stapel, Stringlänge ac <===
17268  866e
17269  866e  20 7e 87  len1   jsr frestr      Textstring übernehmen
17270  8671
17271  8671  a2 00           ldx #0          numerisch
17272  8673  86 0f           stx valtyp      Flag Variablentyp
17273  8675
17274  8675  a8              tay             Stringlänge
17275  8676  60              rts
17276  8677
17277  8677
17278  8677  ===> BASIC-Funktion ASC ($c6) <===
17279  8677
17280  8677  20 6e 86  asc    jsr len1        Stringparam. aus Stapel, Stringlänge
                                             ac
17281  867a  f0 06           beq goasc       Stringlänge 0
17282  867c
17283  867c  a0 00           ldy #0          Zeiger auf 1. Zeichen
17284  867e  20 b7 03         jsr ind1_1      1. Zeichen
17285  8681  a8              tay
17286  8682
17287  8682  4c d4 84  goasc  jmp sngflt      8 Bit INT yr nach GK in FAC
17288  8685
17289  8685
17290  8685  ===> Behandlung »?illegal quantity error« <===
17291  8685
17292  8685  4c 28 7d  gofuc  jmp fcerr       Behandlung »?illegal quantity error«
17293  8688
17294  8688
17295  8688  ===> Arbeitsstring Länge ac anlegen <===
17296  8688
17297  8688  a6 66     strini ldx indice
17298  868a  a4 67            ldy indice+1
17299  868c
17300  868c  86 52           stx dscpnt      ADDRL Zeiger String-Descr.
17301  868e  84 53           sty dscpnt+1    ADDRH Zeiger String-Descr.
17302  8690
17303  8690
17304  8690  ===> Arb.string Länge ac anlegen <===
17305  8690
17306  8690  20 99 92  strspa jsr getspa     Arbeitsstring Länge ac anlegen
17307  8693  86 64           stx dsctmp+1    tmp. Descr.: ADDRL String
17308  8695  84 65           sty dsctmp+2    tmp. Descr.: ADDRH String
```

```
17309  8697  85 63            sta dsctmp       tmp. Descr.: Stringlänge
17310  8699
17311  8699  60                rts
17312  869a
17313  869a
17314  869a  ===> String (Descr.ac/yr) nach Arb.string <===
17315  869a
17316  869a  a2 22       strlit ldx #'"'
17317  869c  86 09              stx charac       Suchzeichen
17318  869e  86 0a              stx endchr       Abbruchzeichen
17319  86a0
17320  86a0
17321  86a0  ===> String übertragen <===
17322  86a0
17323  86a0  85 70       strlt2 sta strng1
17324  86a2  84 71              sty strng1+1
17325  86a4
17326  86a4  85 64              sta dsctmp+1     tmp. Descr.: ADDRL String
17327  86a6  84 65              sty dsctmp+2     tmp. Descr.: ADDRH String
17328  86a8
17329  86a8  a0 ff              ldy #255
17330  86aa  c8          strget iny
17331  86ab  20 f1 42           jsr indst0       LDA (strng1),y aus PGM-RAM
17332  86ae  f0 0c              beq strfi1       Zeilenende
17333  86b0
17334  86b0  c5 09              cmp charac       Suchzeichen
17335  86b2  f0 04              beq strfin
17336  86b4
17337  86b4  c5 0a              cmp endchr       Abbruchzeichen
17338  86b6  d0 f2              bne strget       immer
17339  86b8
17340  86b8
17341  86b8  c9 22       strfin cmp #'"'
17342  86ba  f0 01              beq strfi2
17343  86bc
17344  86bc  18          strfi1 clc
17345  86bd  84 63       strfi2 sty dsctmp       tmp. Descr.: Stringlänge
17346  86bf
17347  86bf  98                 tya              Stringlänge
17348  86c0  65 70              adc strng1
17349  86c2  85 72              sta strng2
17350  86c4
17351  86c4  a6 71              ldx strng1+1
17352  86c6  90 01              bcc strst2
17353  86c8  e8                 inx
17354  86c9  86 73       strst2 stx strng2+1
17355  86cb
17356  86cb  98                 tya              Stringlänge
17357  86cc
17358  86cc
17359  86cc  ===> ac Bytes strng1 ind. nach Arb.String <===
17360  86cc
17361  86cc  20 88 86    strlt1 jsr strini       Arbeitsstring Länge ac anlegen
17362  86cf  a8                 tay              Stringlänge
17363  86d0  f0 11              beq putnew       String-Descriptor in Tabelle
17364  86d2
17365  86d2  48                 pha              Stringlänge
17366  86d3
```

```
17367  86d3  88          strst4 dey              Zeiger in String
17368  86d4  20 f1 42          jsr indst0        LDA (strng1),y aus PGM-RAM
17369  86d7  8d 04 ff          sta lcrd          ROM / VAR-RAM einschalten
17370  86da
17371  86da  91 37              sta (frespc),y    ADDRL Zeiger auf neuen String
17372  86dc
17373  86dc  98                 tya               Zeiger in String
17374  86dd  d0 f4              bne strst4
17375  86df
17376  86df  68                 pla               Stringlänge
17377  86e0  20 71 87           jsr mvdone        ac zu frespc addieren
17378  86e3
17379  86e3
17380  86e3  ===> String-Descriptor in Tabelle <===
17381  86e3
17382  86e3  a6 18       putnew ldx temppt        Zeiger in den String-Stapel
17383  86e5  e0 24              cpx #<index1      Stapel voll
17384  86e7  d0 05              bne putnw1
17385  86e9
17386  86e9  ===> Behandlung »?formula too complex error« <===
17387  86e9
17388  86e9  a2 19              ldx #_erst
17389  86eb  4c 3c 4d           jmp error         ind. Behandlung Fehlermeldung
17390  86ee
17391  86ee
17392  86ee  a5 63       putnw1 lda dsctmp        tmp. Descr.: Stringlänge
17393  86f0  95 00              sta 0,x           in String-Stapel
17394  86f2
17395  86f2  a5 64              lda dsctmp+1      tmp. Descr.: ADDRL String
17396  86f4  95 01              sta 1,x
17397  86f6
17398  86f6  a5 65              lda dsctmp+2      tmp. Descr.: ADDRH String
17399  86f8  95 02              sta 2,x           in String-Stapel17400  86fa
17401  86fa  a0 00              ldy #>tempst      Descriptor-Stapel
17402  86fc  86 66              stx indice        ADDRL Descr. in Stapel
17403  86fe
17404  86fe  84 67              sty indice+1      ADDRH Descr. in Stapel
17405  8700  84 71              sty strng1+1
17406  8702
17407  8702  88                 dey               = $ff: String
17408  8703  84 0f              sty valtyp        Flag Variablentyp
17409  8705
17410  8705  86 19              stx lastpt        ADDRL letzt benutzte String-Adresse
17411  8707
17412  8707  e8                 inx
17413  8708  e8                 inx
17414  8709  e8                 inx
17415  870a  86 18              stx temppt        Zeiger in den String-Stapel
17416  870c
17417  870c  60                 rts
17418  870d
17419  870d
17420  870d  ===> String verketten (+) <===
17421  870d
17422  870d  a5 67       cat    lda indice+1      ADDRH Zeiger Descr. 1. String
17423  870f  48                 pha
17424  8710
17425  8710  a5 66              lda indice        ADDRL Zeiger Descr. 1. String
```

```
17426  8712  48              pha
17427  8713
17428  8713  20 d7 78        jsr eval          ind. folg. Ausdruck auswerten
17429  8716  20 dd 77        jsr chkstr        Test String VAR, sonst type mism.
17430  8719
17431  8719  68              pla               ADDRL Zeiger Descr. 1. String
17432  871a  85 70           sta strng1
17433  871c
17434  871c  68              pla               ADDRH Zeiger Descr. 1. String
17435  871d  85 71           sta strng1+1
17436  871f
17437  871f  a0 00           ldy #0            Zeiger auf Stringlänge 1. String
17438  8721  20 f6 42        jsr indst1        LDA (strng1),y aus VAR-RAM
17439  8724  85 79           sta syntmp        Stringlänge
17440  8726
17441  8726  20 e7 42        jsr indfmo        LDA (facmo),y aus VAR-RAM
17442  8729  18              clc               Stringlänge 2. String
17443  872a  65 79           adc syntmp        + Stringlänge 1. String
17444  872c  90 03           bcc sizeok
17445  872e
17446  872e  4c ed a5        jmp errlen        Behandlung »?string too long«
17447  8731
17448  8731
17449  8731  20 88 86  sizeok jsr strini       Arbeitsstring Länge ac anlegen
17450  8734  20 4e 87        jsr movins        String von Descr. strng1 in
                                                Arb.string kop.
17451  8737
17452  8737  a5 52           lda dscpnt        Zeiger auf 2. Descr. setzen
17453  8739  a4 53           ldy dscpnt+1
17454  873b  20 85 87        jsr fretmp        String ac/yr vom Stapel, ungültig
17455  873e  20 63 87        jsr movdo         String index1 in Arbeitsstring
                                                kopieren
17456  8741
17457  8741  a5 70           lda strng1        Zeiger auf 1. Descr.
17458  8743  a4 71           ldy strng1+1
17459  8745  20 85 87        jsr fretmp        String ac/yr vom Stapel, ungültig
17460  8748  20 e3 86        jsr putnew        String-Descriptor in Tabelle
17461  874b  4c 09 78        jmp tstop
17462  874e
17463  874e
17464  874e  ===> String von Descr. strng1 in Arb.string kop. <===
17465  874e
17466  874e  a0 00      movins ldy #0           Zeiger auf Stringlänge
17467  8750  20 f6 42        jsr indst1        LDA (strng1),y aus VAR-RAM
17468  8753  48              pha               Stringlänge
17469  8754
17470  8754  c8              iny               Zeiger auf ADDRL String
17471  8755  20 f6 42        jsr indst1        LDA (strng1),y aus VAR-RAM
17472  8758  aa              tax               ADDRL String
17473  8759
17474  8759  c8              iny               Zeiger auf ADDRH String
17475  875a  20 f6 42        jsr indst1        LDA (strng1),y aus VAR-RAM
17476  875d  a8              tay               ADDRH String
17477  875e
17478  875e  68              pla               Stringlänge
17479  875f
17480  875f  86 24           stx index1        ADDRL Stringadresse
17481  8761  84 25           sty index1+1      ADDRH Stringadresse
```

```
17482  8763
17483  8763
17484  8763
17484  8763  ===> String index1 in Arbeitsstring kopieren <===
17485  8763
17486  8763  a8          movdo  tay              Stringlänge
17487  8764  f0 0b              beq mvdone       ac zu frespc addieren
17488  8766
17489  8766  48                 pha              Stringlänge
17490  8767
17491  8767  88          movlp  dey              Zeiger in String
17492  8768  20 b7 03           jsr ind1_1       LDA (index1),y aus VAR-RAM
17493  876b  91 37              sta (frespc),y   ADDRL Zeiger auf neuen String
17494  876d
17495  876d  98                 tya              Zeiger in String
17496  876e  d0 f7              bne movlp
17497  8770
17498  8770  68                 pla              Stringlänge
17499  8771
17500  8771
17501  8771  ===> ac zu frespc addieren <===
17502  8771
17503  8771  18          mvdone clc
17504  8772  65 37             adc frespc       ADDRL Zeiger auf neuen String
17505  8774  85 37             sta frespc       ADDRL Zeiger auf neuen String
17506  8776
17507  8776  90 02       mvdon1 bcc mvstrt
17508  8778
17509  8778  e6 38             inc frespc+1     ADDRH Zeiger auf neuen String
17510  877a  60          mvstrt rts
17511  877b
17512  877b
17513  877b  ===> Ausdruck in Arbeitsstring, frestr <===
17514  877b
17515  877b  20 ef 77   sav13  jsr frmevl       folgenden Ausdruck auswerten
17516  877e
17517  877e
17518  877e  ===> Textstring übernehmen <===
17519  877e
17520  877e  20 dd 77   frestr jsr chkstr       Test String VAR, sonst type mism.
17521  8781
17522  8781
17523  8781  ===> Daten Arbeitsstring aus Ablage, Adresse index1, Länge ac <===
17524  8781
17525  8781  a5 66       frefac lda indice      ADDRL Zeiger auf Descr.
17526  8783  a4 67              ldy indice+1     ADDRH Zeiger auf Descr.
17527  8785
17528  8785
17529  8785  ===> String ac/yr vom Stapel, ungültig <===
17530  8785
17531  8785  85 24       fretmp sta index1
17532  8787  84 25              sty index1+1
17533  8789
17534  8789  20 e0 87           jsr fretms       Descriptor vom Stapel entfernen
17535  878c  d0 3c              bne fre02        nicht im Stapel
17536  878e
17537  878e  20 f6 54           jsr stradj       Test Trailer, index1 auf Trailer
17538  8791  90 37              bcc fre02        String ohne Trailer
```

```
17539  8793
17540  8793
17540  8793  ===> String-Trailer ungültig machen <===
17541  8793
17542  8793   index1 zeigt auf Trailer
17543  8793   yr enthält 2#
17544  8793
17545  8793  88                dey               Zeiger letztes Trailerbyte
17546  8794
17547  8794  a9 ff             lda #$ff          Flag für ungültigen String
17548  8796  8d 04 ff          sta lcrd          ROM / VAR-RAM einschalten
17549  8799
17550  8799  91 24            sta (index1),y    in Trailer
17551  879b
17552  879b  88                dey               Zeiger vorletztes Trailerbyte
17553  879c  8a                txa               Stringlänge
17554  879d  91 24            sta (index1),y    in Trailer
17555  879f
17556  879f  ===> Test auf ungült. String Untergrenze Stringbereich <===
17557  879f
17558  879f  48                pha               Stringlänge
17559  87a0
17560  87a0  49 ff            eor #$ff          invertieren
17561  87a2  38                sec               + 1 ergibt negative Stringlänge
17562  87a3  65 24            adc index1
17563  87a5
17564  87a5  a4 25            ldy index1+1
17565  87a7  b0 01            bcs res00
17566  87a9  88                dey
17567  87aa
17568  87aa  85 24      res00  sta index1
17569  87ac  84 25            sty index1+1
17570  87ae
17571  87ae   index1 zeigt auf Stringbeginn
17572  87ae
17573  87ae  aa                tax               ADDRL String
17574  87af
17575  87af  68                pla               Stringlänge
17576  87b0  c4 36            cpy fretop+1      ADDRH Zeiger Anfang String-Variable
17577  87b2  d0 3c            bne frerts        nicht unterster String
17578  87b4
17579  87b4  e4 35            cpx fretop        ADDRL Zeiger Anfang String-Variable
17580  87b6  d0 38            bne frerts        nicht unterster String
17581  87b8
17582  87b8  ===> unterster String ungültig, fretop höher <===
17583  87b8
17584  87b8  48                pha               Stringlänge
17585  87b9
17586  87b9  38                sec
17587  87ba  65 35            adc fretop        ADDRL Zeiger Anfang String-Variable
17588  87bc  85 35            sta fretop        ADDRL Zeiger Anfang String-Variable
17589  87be
17590  87be  90 02            bcc fre01
17591  87c0  e6 36            inc fretop+1      ADDRH Zeiger Anfang String-Variable
17592  87c2
17593  87c2  e6 35      fre01 inc fretop        zweites Trailerbyte
17594  87c4  d0 02            bne frepla
17595  87c6  e6 36            inc fretop+1      ADDRH Zeiger Anfang String-Variable
```

```
17596  87c8
17597  87c8  68          frepla pla              Stringlänge
17598  87c9  60                 rts
17599  87ca
17600  87ca
17601  87ca  ===> Stringadresse nach index1, Länge ac <===
17602  87ca
17603  87ca  a0 00       fre02  ldy #0            Zeiger auf Stringlänge
17604  87cc  20 b7 03           jsr ind1_1        LDA (index1),y aus VAR-RAM
17605  87cf  48                 pha               Stringlänge
17606  87d0
17607  87d0  c8                 iny               Zeiger auf ADDRL String
17608  87d1  20 b7 03           jsr ind1_1        LDA (index1),y aus VAR-RAM
17609  87d4  aa                 tax               ADDRL String
17610  87d5
17611  87d5  c8                 iny               Zeiger auf ADDRH String
17612  87d6  20 b7 03           jsr ind1_1        LDA (index1),y aus VAR-RAM
17613  87d9  a8                 tay               ADDRH Stringadresse
17614  87da
17615  87da  86 24              stx index1        ADDRL Stringadresse
17616  87dc  84 25              sty index1+1      ADDRH Stringadresse
17617  87de  68                 pla               Stringlänge
17618  87df
17619  87df  60                 rts
17620  87e0
17621  87e0
17622  87e0  ===> Descriptor vom Stapel entfernen <===
17623  87e0
17624  87e0  c4 1a       fretms cpy lastpt+1      ADDRH letzt benutzte String-Adresse
17625  87e2  d0 0c              bne frerts
17626  87e4
17627  87e4  c5 19              cmp lastpt        ADDRL letzt benutzte String-Adresse
17628  87e6  d0 08              bne frerts        Descr. nicht im Stapel
17629  87e8
17630  87e8  85 18              sta temppt        Zeiger in den String-Stapel
17631  87ea
17632  87ea  e9 03              sbc #3            Descr. Länge
17633  87ec  85 19              sta lastpt        ADDRL letzt benutzte String-Adresse
17634  87ee
17635  87ee  a0 00              ldy #0            ZERO =1
17636  87f0
17637  87f0  60         frerts rts
17638  87f1
17639  87f1
17640  87f1                    .end
17641  87f1                    .lib basic.math_2
17642  87f1  put"@basic.math_2"
```

```
17644  87f1
17645  87f1
17646  87f1  ===> chrget, 8 Bit INT nach xr <===
17647  87f1
17648  87f1  20 80 03   gtbytc jsr chrget       nächstes Zeichen nach ac
17649  87f4
17650  87f4
17651  87f4  ===> 8 Bit INT üb. frmnum nach xr <===
17652  87f4
17653  87f4  20 d7 77   getbyt jsr frmnum       folg. num. Variable nach FAC
17654  87f7
17655  87f7
17656  87f7  ===> 8 Bit INT CONST aus FAC nach xr <===
17657  87f7
17658  87f7  20 ad 84   conint jsr posint       Wandlung FAC nach pos. INT
17659  87fa
17660  87fa  a6 66             ldx facmo        High-Byte Int.
17661  87fc  d0 2d             bne gofcer       Behandlung »?illegal quantity error«
17662  87fe
17663  87fe  a6 67             ldx faclo        Low-Byte Int.
17664  8800  4c 86 03          jmp chrgot       letztes Zeichen erneut nach ac
17665  8803
17666  8803
17667  8803  ===> 16 Bit INT nach linnum, Kommatest, 8 Bit INT nach xr <===
17668  8803
17669  8803  20 d7 77   getnum jsr frmnum       folg. num. Variable nach FAC
17670  8806  20 15 88          jsr getadr       pos. FAC als 16 Bit INT linnum,
                                                yr/ac
17671  8809
17672  8809
17673  8809  ===> Kommatest, 8 Bit INT nach xr <===
17674  8809
17675  8809  20 5c 79   combyt jsr chkcom       Test auf folg. »Komma« sonst Fehler
17676  880c  4c f4 87          jmp getbyt       8 Bit INT üb. frmnum nach xr
17677  880f
17678  880f
17679  880f  ===> Kommatest, 16 Bit INT nach linnum <===
17680  880f
17681  880f  20 5c 79   comwrd jsr chkcom       Test auf folg. »Komma« sonst Fehler
17682  8812
17683  8812
17684  8812  ===> 16 Bit INT nach linnum <===
17685  8812
17686  8812  20 d7 77   getwrd jsr frmnum       folg. num. Variable nach FAC
17687  8815
17688  8815
17689  8815  ===> pos. FAC als 16 Bit INT linnum, yr/ac <===
17690  8815
17691  8815  a5 68      getadr lda facsgn       FAC: SG
17692  8817  30 12             bmi gofcer       negativ
17693  8819
17694  8819
17695  8819  ===> FAC als 16 Bit INT linnum, yr/ac <===
17696  8819
17697  8819  a5 63      getsad lda facexp       FAC: EX
17698  881b  c9 91             cmp #_expcn+16
17699  881d  b0 0c             bcs gofcer       Exponent >15
17700  881f
```

```
17701  881f  20 c7 8c          jsr qint          FAC GK nach INT
17702  8822
17703  8822  a5 66             lda facmo         FAC: M2
17704  8824  a4 67             ldy faclo         FAC: M1
17705  8826
17706  8826  84 16             sty linnum
17707  8828  85 17             sta linnum+1
17708  882a
17709  882a  60                rts
17710  882b
17711  882b  ===> Behandlung »?illegal quantity error« <===
17712  882b
17713  882b  4c 28 7d   gofcer jmp fcerr        Behandlung »?illegal quantity error«
17714  882e
17715  882e
17716  882e  ===> RAM Var. ac/yr minus FAC <===
17717  882e
17718  882e  20 b4 8a   fsub   jsr conupk       RAM Var. ac/yr nach ARG
17719  8831
17720  8831
17721  8831  ===> BASIC-Funktion MINUS ($ab) <===
17722  8831
17723  8831  a5 68      fsubt  lda facsgn        FAC: SG
17724  8833  49 ff             eor #$ff          invertieren (MINUS Operator)
17725  8835  85 68             sta facsgn        FAC: SG
17726  8837
17727  8837  45 6f             eor argsgn        ARG: SG
17728  8839  85 70             sta arisgn        Vorzeichen Ergebnis Operation
17729  883b
17730  883b  a5 63             lda facexp        FAC: EX
17731  883d  4c 48 88          jmp faddt         BASIC-Funktion PLUS ($aa)
17732  8840
17733  8840
17734  8840  ===> EX angleichen, FAC plus VAR <===
17735  8840
17736  8840  20 79 89   fadd5  jsr shiftr        FAC rechts schieben bis EXP = 0
17737  8843  90 3d             bcc fadd4         immer -> Sprung
17738  8845
17739  8845
17740  8845  ===> FAC plus Var. ac/yr <===
17741  8845
17742  8845  20 b4 8a   fadd   jsr conupk       RAM Var. ac/yr nach ARG
17743  8848
17744  8848
17745  8848  ===> BASIC-Funktion PLUS ($aa) <===
17746  8848
17747  8848  d0 03      faddt  bne faddb
17748  884a
17749  884a  ===> FAC = 0 <===
17750  884a
17751  884a  4c 28 8c          jmp movfa         ARG nach FAC
17752  884d
17753  884d
17754  884d  a6 71      faddb  ldx facov         Rundungsbyte
17755  884f  86 58             stx oldov
17756  8851
17757  8851  a2 6a             ldx #<argexp      Zeiger auf ARG
17758  8853  a5 6a             lda argexp        ARG: EX
```

```
17759  8855
17760  8855
17761  8855  ===> FAC plus ARG <===
17762  8855
17763  8855  a8          faddc  tay
17764  8856  d0 01              bne fadde
17765  8858
17766  8858  ===> ARG = 0 <===
17767  8858
17768  8858  60                 rts
17769  8859
17770  8859
17771  8859  38          fadde  sec
17772  885a  e5 63              sbc facexp       FAC: EX
17773  885c  f0 24              beq fadd4        EX FAC = EX ARG
17774  885e  90 12              bcc fadda        EX FAC > EX ARG
17775  8860
17776  8860  ===> EX FAC < EX ARG <===
17777  8860
17778  8860  84 63              sty facexp       FAC: EX
17779  8862
17780  8862  a4 6f              ldy argsgn       ARG: SG
17781  8864  84 68              sty facsgn       FAC: SG
17782  8866
17783  8866  49 ff              eor #$ff         Differenz EX ARG - EX FAC
                                                invertieren
17784  8868  69 00              adc #0           + 1
17785  886a
17786  886a  a0 00              ldy #0
17787  886c  84 58              sty oldov
17788  886e
17789  886e  a2 63              ldx #<facexp     Zeiger auf FAC
17790  8870  d0 04              bne fadd1        immer
17791  8872
17792  8872  a0 00       fadda  ldy #0
17793  8874  84 71              sty facov        Rundungsbyte
17794  8876
17795  8876  c9 f9       fadd1  cmp #256-7
17796  8878  30 c6              bmi fadd5        EX Differenz > 7
17797  887a
17798  887a  a8                 tay
17799  887b  a5 71              lda facov        Rundungsbyte
17800  887d  56 01              lsr 0+_m4,x
17801  887f  20 90 89           jsr rolshf       gleichmachen
17802  8882
17803  8882  24 70       fadd4  bit arisgn       Vorzeichen Ergebnis Operation
17804  8884  10 57              bpl fadd2        Vorzeichen gleich
17805  8886
17806  8886  a0 63              ldy #<facexp     Zeiger auf FAC
17807  8888  e0 6a              cpx #<argexp     Zeiger auf ARG
17808  888a  f0 02              beq subit        Mantissen subtrahieren
17809  888c
17810  888c  a0 6a              ldy #<argexp     Zeiger auf ARG
17811  888e
17812  888e
17813  888e  ===> Mantissen subtrahieren <===
17814  888e
17815  888e  38          subit  sec
```

```
17816  888f  49 ff           eor #$ff
17817  8891  65 58           adc oldov
17818  8893  85 71           sta facov       Rundungsbyte
17819  8895
17820  8895  b9 04 00        lda 0+_m1,y
17821  8898  f5 04           sbc 0+_m1,x
17822  889a  85 67           sta faclo       FAC: M1
17823  889c
17824  889c  b9 03 00        lda 0+_m2,y
17825  889f  f5 03           sbc 0+_m2,x
17826  88a1  85 66           sta facmo       FAC: M2
17827  88a3
17828  88a3  b9 02 00        lda 0+_m3,y
17829  88a6  f5 02           sbc 0+_m3,x
17830  88a8  85 65           sta facmoh      FAC: M3
17831  88aa
17832  88aa  b9 01 00        lda 0+_m4,y
17833  88ad  f5 01           sbc 0+_m4,x
17834  88af  85 64           sta facho       FAC: M4
17835  88b1
17836  88b1
17837  88b1  ===> Test/SG invert., FAC normalisieren <===
17838  88b1
17839  88b1  b0 03     fadflt bcs normal      FAC normalisieren
17840  88b3
17841  88b3  20 26 89         jsr negfac      FAC einschl. SGN invertieren
17842  88b6
17843  88b6
17844  88b6  ===> FAC normalisieren <===
17845  88b6
17846  88b6  a0 00     normal ldy #$00
17847  88b8  98               tya             Zähler für Bitverschiebung
17848  88b9
17849  88b9  18               clc
17850  88ba  a6 64     norm3  ldx facho       FAC: M4
17851  88bc  d0 4a            bne norm2
17852  88be
17853  88be  ===> FAC byteweise links schieben <===
17854  88be
17855  88be  a6 65            ldx facmoh      FAC: M3
17856  88c0  86 64            stx facho       FAC: M4
17857  88c2
17858  88c2  a6 66            ldx facmo       FAC: M2
17859  88c4  86 65            stx facmoh      FAC: M3
17860  88c6
17861  88c6  a6 67            ldx faclo       FAC: M1
17862  88c8  86 66            stx facmo       FAC: M2
17863  88ca
17864  88ca  a6 71            ldx facov       Rundungsbyte
17865  88cc  86 67            stx faclo       FAC: M1
17866  88ce
17867  88ce  84 71            sty facov       Rundungsbyte
17868  88d0
17869  88d0  69 08            adc #8          Verschiebung um 8 Bits
17870  88d2  c9 20            cmp #4*8        4 Bytes
17871  88d4  d0 e4            bne norm3
17872  88d6
17873  88d6
```

```
17874  88d6
17874  88d6  ===> FAC auf 00 setzen <===
17875  88d6
17876  88d6  a9 00      zerofc lda #$00
17877  88d8
17878  88d8  85 63      zerof1 sta facexp       FAC: EX
17879  88da
17880  88da  85 68      zeroml sta facsgn       FAC: SG
17881  88dc
17882  88dc  60               rts
17883  88dd
17884  88dd
17885  88dd  ===> Mantissen-Addition bei gleichen Vorzeichen <===
17886  88dd
17887  88dd  65 58      fadd2  adc oldov
17888  88df  85 71             sta facov       Rundungsbyte
17889  88e1
17890  88e1  a5 67             lda faclo       FAC: M1
17891  88e3  65 6e             adc arglo       ARG: M1
17892  88e5  85 67             sta faclo       FAC: M1
17893  88e7
17894  88e7  a5 66             lda facmo       FAC: M2
17895  88e9  65 6d             adc argmo       ARG: M2
17896  88eb  85 66             sta facmo       FAC: M2
17897  88ed
17898  88ed  a5 65             lda facmoh      FAC: M3
17899  88ef  65 6c             adc argmoh      ARG: M3
17900  88f1  85 65             sta facmoh      FAC: M3
17901  88f3
17902  88f3  a5 64             lda facho       FAC: M4
17903  88f5  65 6b             adc argho       ARG: M4
17904  88f7  85 64             sta facho       FAC: M4
17905  88f9
17906  88f9  4c 15 89          jmp squeez      Test FAC Überlauf, 1 Bit nach rechts
17907  88fc
17908  88fc
17909  88fc  ===> FAC bitweise links schieben <===
17910  88fc
17911  88fc  69 01      norm1  adc #1
17912  88fe  06 71             asl facov       Rundungsbyte
17913  8900  26 67             rol faclo       FAC: M1
17914  8902  26 66             rol facmo       FAC: M2
17915  8904  26 65             rol facmoh      FAC: M3
17916  8906  26 64             rol facho       FAC: M4
17917  8908  10 f2      norm2  bpl norm1       FAC bitweise links schieben
17918  890a
17919  890a  ===> EXP korrigieren <===
17920  890a
17921  890a  38               sec
17922  890b  e5 63             sbc facexp      FAC: EX
17923  890d  b0 c7             bcs zerofc      FAC auf 00 setzen
17924  890f
17925  890f  49 ff             eor #$ff
17926  8911  69 01             adc #1
17927  8913  85 63             sta facexp      FAC: EX
17928  8915
17929  8915
17930  8915
```

```
17930  8915  ===> Test FAC Überlauf, 1 Bit nach rechts <===
17931  8915
17932  8915  90 0e      squeez bcc rndrts       kein Überlauf
17933  8917
17934  8917
17935  8917  ===> Überlaufstelle einschieben <===
17936  8917
17937  8917  e6 63      rndshf inc facexp       FAC: EX
17938  8919  f0 42             beq overr        Behandlung »?overflow error«
17939  891b
17940  891b  ===> FAC 1 Bit rechts schieben <===
17941  891b
17942  891b  66 64             ror facho        FAC: M4
17943  891d  66 65             ror facmoh       FAC: M3
17944  891f  66 66             ror facmo        FAC: M2
17945  8921  66 67             ror faclo        FAC: M1
17946  8923  66 71             ror facov        Rundungsbyte
17947  8925  60        rndrts rts
17948  8926
17949  8926
17950  8926  ===> FAC einschl. SGN invertieren <===
17951  8926
17952  8926  a5 68      negfac lda facsgn       FAC: SG
17953  8928  49 ff             eor #$ff
17954  892a  85 68             sta facsgn       FAC: SG
17955  892c
17956  892c
17957  892c  ===> FAC invertieren <===
17958  892c
17959  892c  a5 64      negfch lda facho        FAC: M4
17960  892e  49 ff             eor #$ff
17961  8930  85 64             sta facho        FAC: M4
17962  8932
17963  8932  a5 65             lda facmoh       FAC: M3
17964  8934  49 ff             eor #$ff
17965  8936  85 65             sta facmoh       FAC: M3
17966  8938
17967  8938  a5 66             lda facmo        FAC: M2
17968  893a  49 ff             eor #$ff
17969  893c  85 66             sta facmo        FAC: M2
17970  893e
17971  893e  a5 67             lda faclo        FAC: M1
17972  8940  49 ff             eor #$ff
17973  8942  85 67             sta faclo        FAC: M1
17974  8944
17975  8944  a5 71             lda facov        Rundungsbyte
17976  8946  49 ff             eor #$ff
17977  8948  85 71             sta facov        Rundungsbyte
17978  894a
17979  894a  e6 71             inc facov        Rundungsbyte
17980  894c  d0 0e             bne incfrt
17981  894e
17982  894e
17983  894e  ===> Mantisse FAC inc. (aufrunden) <===
17984  894e
17985  894e  e6 67      incfac inc faclo        FAC: M1
17986  8950  d0 0a             bne incfrt
17987  8952
```

```
17988  8952  e6 66          inc facmo        FAC: M2
17989  8954  d0 06          bne incfrt
17990  8956
17991  8956  e6 65          inc facmoh       FAC: M3
17992  8958  d0 02          bne incfrt
17993  895a
17994  895a  e6 64          inc facho        FAC: M4
17995  895c  60       incfrt rts
17996  895d
17997  895d
17998  895d  ===> Behandlung »?overflow error« <===
17999  895d
18000  895d  a2 0f     overr  ldx #_erov
18001  895f  4c 3c 4d         jmp error        ind. Behandlung Fehlermeldung
18002  8962
18003  8962
18004  8962  ===> RES  <===
18005  8962
18006  8962  a2 27     mulshf ldx #<resho-1    RES: M4
18007  8964
18008  8964
18009  8964  ===> GK-ACCU, Offs. xr, rechts schieben <===
18010  8964
18011  8964  b4 04     shftr2 ldy 0+_m1,x
18012  8966  84 71            sty facov        Rundungsbyte
18013  8968
18014  8968  b4 03            ldy 0+_m2,x
18015  896a  94 04            sty 0+_m1,x
18016  896c
18017  896c  b4 02            ldy 0+_m3,x
18018  896e  94 03            sty 0+_m2,x
18019  8970
18020  8970  b4 01            ldy 0+_m4,x
18021  8972  94 02            sty 0+_m3,x
18022  8974
18023  8974  ac df 03         ldy bits         FAC Overflow Bit
18024  8977  94 01            sty 0+_m4,x
18025  8979
18026  8979
18027  8979  ===> FAC rechts schieben bis EXP = 0 <===
18028  8979
18029  8979  69 08     shiftr adc #8           Bit-Zähler
18030  897b  30 e7            bmi shftr2       GK-ACCU, Offs. xr, rechts schieben
18031  897d  f0 e5            beq shftr2       GK-ACCU, Offs. xr, rechts schieben
18032  897f
18033  897f  e9 08            sbc #8           Bit-Zähler
18034  8981  a8               tay
18035  8982
18036  8982  a5 71            lda facov        Rundungsbyte
18037  8984  b0 14            bcs shftrt
18038  8986
18039  8986  16 01     shftr3 asl 0+_m4,x
18040  8988  90 02            bcc shftr4
18041  898a
18042  898a  f6 01            inc 0+_m4,x
18043  898c
18044  898c  76 01     shftr4 ror 0+_m4,x
18045  898e  76 01            ror 0+_m4,x
```

```
18046  8990
18047  8990
18048  8990  ===> FAC bitweise nach rechts schieben <===
18049  8990
18050  8990  76 02       rolshf ror 0+_m3,x
18051  8992  76 03              ror 0+_m2,x
18052  8994  76 04              ror 0+_m1,x
18053  8996  6a                ror a
18054  8997
18055  8997  c8                iny              Bit-Zähler
18056  8998  d0 ec             bne shftr3
18057  899a
18058  899a  18          shftrt clc
18059  899b  60                rts
18060  899c
18061  899c
18062  899c  ===> GK-Konstante 1.0 <===
18063  899c
18064  899c  81 00 00    fone   .byte $81, $00, $00, $00, $00
18064  899f  00 00
18065  89a1
18066  89a1
18067  89a1  ===> Grad (3) Polynom für LOG <===
18068  89a1
18069  89a1  03          logcn2 .byte $03        Polynomgrad
18070  89a2
18071  89a2
18072  89a2  ===> Koeffizient 3: 0.434 255 942 <===
18073  89a2
18074  89a2  7f 5e 56    logco3 .byte $7f, $5e, $56, $cb, $79
18074  89a5  cb 79
18075  89a7
18076  89a7
18077  89a7  ===> Koeffizient 2: 0.576 584 541 <===
18078  89a7
18079  89a7  80 13 9b    logco2 .byte $80, $13, $9b, $0b, $64
18079  89aa  0b 64
18080  89ac
18081  89ac
18082  89ac  ===> Koeffizient 1: 0.961 800 759 <===
18083  89ac
18084  89ac  80 76 38    logco1 .byte $80, $76, $38, $93, $16
18084  89af  93 16
18085  89b1
18086  89b1
18087  89b1  ===> Koeffizient 0: 2.885 390 07 <===
18088  89b1
18089  89b1  82 38 aa    logco0 .byte $82, $38, $aa, $3b, $20
18089  89b4  3b 20
18090  89b6
18091  89b6
18092  89b6  ===> GK-Konstante 0.707 106 781 = 1/2 SQR 2 <===
18093  89b6
18094  89b6  80 35 04    sqr05  .byte $80, $35, $04, $f3, $34
18094  89b9  f3 34
18095  89bb
18096  89bb
18097  89bb
```

```
18097  89bb  ===> GK-Konstante 1.414 213  56 = SQR 2 <===
18098  89bb
18099  89bb  81 35 04   sqr20  .byte $81, $35, $04, $f3, $34
18099  89be  f3 34
18100  89c0
18101  89c0
18102  89c0  ===> GK-Konstante -0.5 <===
18103  89c0
18104  89c0  80 80 00   neghlf .byte $80, $80, $00, $00, $00
18104  89c3  00 00
18105  89c5
18106  89c5
18107  89c5  ===> GK-Konstante ln 2 <===
18108  89c5
18109  89c5  80 31 72   log2   .byte $80, $31, $72, $17, $f8
18109  89c8  17 f8
18110  89ca
18111  89ca
18112  89ca  ===> BASIC-Funktion LOG ($bc) <===
18113  89ca
18114  89ca  20 57 8c   log    jsr sign         Flag für FAC NULL, PLUS, MINUS nach
                                                ac
18115  89cd  f0 02             beq logerr       FAC NULL
18116  89cf  10 03             bpl log1         FAC PLUS
18117  89d1
18118  89d1  ===> Behandlung »?illegal quantity error« <===18119  89d1
18120  89d1  4c 28 7d   logerr jmp fcerr        Behandlung »?illegal quantity error«
18121  89d4
18122  89d4
18123  89d4  a5 63      log1   lda facexp       FAC: EX
18124  89d6  e9 7f             sbc #_expcn-2    CARRY=0
18125  89d8  48               pha              Transf. Exponent
18126  89d9
18127  89d9  a9 80             lda #_expcn-1
18128  89db  85 63             sta facexp       FAC: EX
18129  89dd
18130  89dd  a9 b6             lda #<sqr05      GK-Konstante 0.707 106 781 = 1/2 SQR
                                                2
18131  89df  a0 89             ldy #>sqr05      GK-Konstante 0.707 106 781 = 1/2 SQR
                                                2
18132  89e1  20 12 8a          jsr romadd       ROM Konst. ac/yr zu FAC addieren
18133  89e4
18134  89e4  a9 bb             lda #<sqr20      GK-Konstante 1.414 213  56 = SQR 2
18135  89e6  a0 89             ldy #>sqr20      GK-Konstante 1.414 213  56 = SQR 2
18136  89e8  20 1e 8a          jsr romdiv       ROM Konst. ac/yr durch FAC div.
18137  89eb
18138  89eb  a9 9c             lda #<fone       GK-Konstante 1.0
18139  89ed  a0 89             ldy #>fone       GK-Konstante 1.0
18140  89ef  20 18 8a          jsr romsub       ROM Konst. ac/yr von FAC subtr.
18141  89f2
18142  89f2  a9 a1             lda #<logcn2     Grad (3) Polynom für LOG
18143  89f4  a0 89             ldy #>logcn2     Grad (3) Polynom für LOG
18144  89f6  20 86 90          jsr polyx        Polynom-Auswertung
18145  89f9
18146  89f9  a9 c0             lda #<neghlf     GK-Konstante -0.5
18147  89fb  a0 89             ldy #>neghlf     GK-Konstante -0.5
18148  89fd  20 12 8a          jsr romadd       ROM Konst. ac/yr zu FAC addieren
18149  8a00
```

```
18150  8a00  68              pla             Transf. Exponent
18151  8a01  20 b0 8d        jsr finlog      FAC plus INT in ac
18152  8a04
18153  8a04  a9 c5           lda #<log2      GK-Konstante ln 2
18154  8a06  a0 89           ldy #>log2      GK-Konstante ln 2
18155  8a08
18156  8a08
18157  8a08  ===> ROM Konst. ac/yr mit FAC mult. <===
18158  8a08
18159  8a08  20 89 8a   rommlt jsr romupk     ROM Konst. ac/yr nach ARG
18160  8a0b
18161  8a0b  4c 27 8a         jmp fmultt      BASIC-Funktion MULTIPLIKATION ($ac)
18162  8a0e
18163  8a0e
18164  8a0e  ===> 0.5 zu FAC addieren <===
18165  8a0e
18166  8a0e  a9 76      faddh  lda #<fhalf     GK-Konstante 0.5 für SQR
18167  8a10  a0 8f           ldy #>fhalf     GK-Konstante 0.5 für SQR
18168  8a12
18169  8a12
18170  8a12  ===> ROM Konst. ac/yr zu FAC addieren <===
18171  8a12
18172  8a12  20 89 8a   romadd jsr romupk     ROM Konst. ac/yr nach ARG
18173  8a15
18174  8a15  4c 48 88         jmp faddt       BASIC-Funktion PLUS ($aa)
18175  8a18
18176  8a18
18177  8a18  ===> ROM Konst. ac/yr von FAC subtr. <===
18178  8a18
18179  8a18  20 89 8a   romsub jsr romupk     ROM Konst. ac/yr nach ARG
18180  8a1b  4c 31 88         jmp fsubt       BASIC-Funktion MINUS ($ab)
18181  8a1e
18182  8a1e
18183  8a1e  ===> ROM Konst. ac/yr durch FAC div. <===
18184  8a1e
18185  8a1e  20 89 8a   romdiv jsr romupk     ROM Konst. ac/yr nach ARG
18186  8a21  4c 4c 8b         jmp fdivt       BASIC-Funktion DIVISION / ($ad)
18187  8a24
18188  8a24
18189  8a24  ===> FAC mal RAM VAR ac/yr <===
18190  8a24
18191  8a24  20 b4 8a   fmult  jsr conupk     RAM Var. ac/yr nach ARG
18192  8a27
18193  8a27
18194  8a27  ===> BASIC-Funktion MULTIPLIKATION ($ac) <===
18195  8a27
18196  8a27  d0 03      fmultt bne fmult1
18197  8a29
18198  8a29  ===> Multiplikation mit NULL <===
18199  8a29
18200  8a29  4c 88 8a         jmp multrt
18201  8a2c
18202  8a2c
18203  8a2c  20 ec 8a   fmult1 jsr muldiv      ARG EX plus FAC EX
18204  8a2f
18205  8a2f  a9 00            lda #$00
18206  8a31  85 28            sta resho       RES: M4
18207  8a33  85 29            sta resmoh      RES: M3
```

```
18208  8a35  85 2a           sta resmo        RES: M
18209  8a37  85 2b           sta reslo        RES: M1
18210  8a39
18211  8a39  a5 71           lda facov        Rundungsbyte
18212  8a3b  20 55 8a        jsr mltply       ARG mal ac(FAC) RES
18213  8a3e
18214  8a3e  a5 67           lda faclo        FAC: M1
18215  8a40  20 55 8a        jsr mltply       ARG mal ac(FAC) RES
18216  8a43
18217  8a43  a5 66           lda facmo        FAC: M2
18218  8a45  20 55 8a        jsr mltply       ARG mal ac(FAC) RES
18219  8a48
18220  8a48  a5 65           lda facmoh       FAC: M3
18221  8a4a  20 55 8a        jsr mltply       ARG mal ac(FAC) RES
18222  8a4d
18223  8a4d  a5 64           lda facho        FAC: M4
18224  8a4f  20 5b 8a        jsr mltpl1
18225  8a52
18226  8a52  4c c1 8b        jmp movfr        RES nach FAC, normalisieren
18227  8a55
18228  8a55
18229  8a55  ===> ARG mal ac(FAC) RES <===
18230  8a55
18231  8a55  d0 04      mltply bne mltpl1
18232  8a57
18233  8a57  ===> ac = 0, 8 Bit rechts schieben <===
18234  8a57
18235  8a57  38               sec
18236  8a58  4c 62 89        jmp mulshf       RES
18237  8a5b
18238  8a5b
18239  8a5b  4a         mltpl1 lsr a
18240  8a5c  09 80             ora #%10000000   Bit7 nachschieben
18241  8a5e
18242  8a5e  a8         mltpl2 tay
18243  8a5f  90 19             bcc mltpl3       RES 1 Bit nach rechts
18244  8a61
18245  8a61  ===> RES + ARG nach RES <===
18246  8a61
18247  8a61  18               clc
18248  8a62  a5 2b            lda reslo        RES: M1
18249  8a64  65 6e            adc arglo        ARG: M1
18250  8a66  85 2b            sta reslo        RES: M1
18251  8a68
18252  8a68  a5 2a            lda resmo        RES: M2
18253  8a6a  65 6d            adc argmo        ARG: M2
18254  8a6c  85 2a            sta resmo        RES: M2
18255  8a6e
18256  8a6e  a5 29            lda resmoh       RES: M3
18257  8a70  65 6c            adc argmoh       ARG: M3
18258  8a72  85 29            sta resmoh       RES: M3
18259  8a74
18260  8a74  a5 28            lda resho        RES: M4
18261  8a76  65 6b            adc argho        ARG: M4
18262  8a78  85 28            sta resho        RES: M4
18263  8a7a
18264  8a7a
18265  8a7a
```

```
18265  8a7a  ===> RES 1 Bit nach rechts <===
18266  8a7a
18267  8a7a  66 28      mltpl3 ror resho        RES: M4
18268  8a7c  66 29             ror resmoh       RES: M3
18269  8a7e  66 2a             ror resmo        RES: M2
18270  8a80  66 2b             ror reslo        RES: M1
18271  8a82  66 71             ror facov        Rundungsbyte
18272  8a84
18273  8a84  98                tya              Rest von ac-Byte
18274  8a85  4a                lsr a
18275  8a86  d0 d6             bne mltpl2
18276  8a88
18277  8a88  60        multrt rts
18278  8a89
18279  8a89
18280  8a89  ===> ROM Konst. ac/yr nach ARG <===
18281  8a89
18282  8a89  85 24      romupk sta index1        addrl Quelladresse
18283  8a8b
18284  8a8b  84 25             sty index1+1     addrh Quelladresse
18285  8a8d
18286  8a8d  a0 04             ldy #_m1         Zeiger auf M1
18287  8a8f  b1 24             lda (index1),y   M1
18288  8a91  85 6e             sta arglo        ARG: M1
18289  8a93
18290  8a93  88                dey
18291  8a94  b1 24             lda (index1),y   M2
18292  8a96  85 6d             sta argmo        ARG: M2
18293  8a98
18294  8a98  88                dey
18295  8a99  b1 24             lda (index1),y   M3
18296  8a9b  85 6c             sta argmoh       ARG: M3
18297  8a9d
18298  8a9d  88                dey
18299  8a9e  b1 24             lda (index1),y   M4
18300  8aa0  85 6f             sta argsgn       ARG: SG
18301  8aa2
18302  8aa2  45 68             eor facsgn       FAC: SG
18303  8aa4  85 70             sta arisgn       Vorzeichen Ergebnis Operation
18304  8aa6
18305  8aa6  a5 6f             lda argsgn       ARG: SG
18306  8aa8  09 80             ora #%10000000
18307  8aaa  85 6b             sta argho        ARG: M4
18308  8aac
18309  8aac  88                dey
18310  8aad  b1 24             lda (index1),y   EX
18311  8aaf  85 6a             sta argexp       ARG: EX
18312  8ab1
18313  8ab1  a5 63             lda facexp       FAC: EX
18314  8ab3  60                rts
18315  8ab4
18316  8ab4
18317  8ab4  ===> RAM Var. ac/yr nach ARG <===
18318  8ab4
18319  8ab4  85 24      conupk sta index1        ADDRL
18320  8ab6  84 25             sty index1+1     ADDRH
18321  8ab8
18322  8ab8  ad 00 ff          lda mmucr        MMU 8722 Config. Reg. (Kopie)
```

```
18323  8abb  48              ph
18324  8abc
18325  8abc  a0 04           ldy #_m1          Zeiger in Variable
18326  8abe  20 b7 03        jsr ind1_1        M1
18327  8ac1  85 6e           sta arglo         ARG: M1
18328  8ac3
18329  8ac3  88              dey
18330  8ac4  20 b7 03        jsr ind1_1        M2
18331  8ac7  85 6d           sta argmo         ARG: M2
18332  8ac9
18333  8ac9  88              dey
18334  8aca  20 b7 03        jsr ind1_1        M3
18335  8acd  85 6c           sta argmoh        ARG: M3
18336  8acf
18337  8acf  88              dey
18338  8ad0  20 b7 03        jsr ind1_1        M4
18339  8ad3  85 6f           sta argsgn        ARG: SG
18340  8ad5
18341  8ad5  45 68           eor facsgn        FAC: SG
18342  8ad7  85 70           sta arisgn        Vorzeichen Ergebnis Operation
18343  8ad9
18344  8ad9  a5 6f           lda argsgn        ARG: SG
18345  8adb  09 80           ora #%10000000
18346  8add  85 6b           sta argho         ARG: M4
18347  8adf
18348  8adf  88              dey
18349  8ae0  20 b7 03        jsr ind1_1        EX
18350  8ae3  85 6a           sta argexp        ARG: EX
18351  8ae5
18352  8ae5  68              pla
18353  8ae6  8d 00 ff        sta mmucr         MMU 8722 Config. Reg. (Kopie)
18354  8ae9
18355  8ae9  a5 63           lda facexp        FAC: EX
18356  8aeb  60              rts
18357  8aec
18358  8aec
18359  8aec                  .end
18360  8aec                  .lib basic.math_3
18361  8aec  put@basic.math_3
```

```
18363  8aec
18364  8aec
18365  8aec  ===> ARG EX plus FAC EX <===
18366  8aec
18367  8aec  a5 6a      muldiv lda argexp       ARG: EX
18368  8aee
18369  8aee
18370  8aee  ===> ac EX plus FAC EX <===
18371  8aee
18372  8aee  f0 1f      mldexp beq zeremv
18373  8af0
18374  8af0  18               clc
18375  8af1  65 63            adc facexp       FAC: EX
18376  8af3  90 04            bcc tryoff
18377  8af5
18378  8af5  30 1d            bmi goover       Behandlung »?overflow error«
18379  8af7
18380  8af7  18               clc
18381  8af8  2c               .byte _skip2
18382  8af9
18383  8af9  10 14      tryoff bpl zeremv
18384  8afb
18385  8afb  69 80            adc #_expcn-1
18386  8afd  85 63            sta facexp       FAC: EX
18387  8aff
18388  8aff  d0 03            bne tryof1
18389  8b01
18390  8b01  4c da 88         jmp zeroml
18391  8b04
18392  8b04
18393  8b04  a5 70      tryof1 lda arisgn       Vorzeichen Ergebnis Operation
18394  8b06  85 68            sta facsgn       FAC: SG
18395  8b08
18396  8b08  60               rts
18397  8b09
18398  8b09
18399  8b09  a5 68      mldvex lda facsgn       FAC: SG
18400  8b0b  49 ff            eor #$ff
18401  8b0d  30 05            bmi goover       Behandlung »?overflow error«
18402  8b0f
18403  8b0f  68         zeremv pla
18404  8b10  68               pla
18405  8b11  4c d6 88         jmp zerofc       FAC auf 00 setzen
18406  8b14
18407  8b14
18408  8b14  ===> Behandlung »?overflow error« <===
18409  8b14
18410  8b14  4c 5d 89   goover jmp overr        Behandlung »?overflow error«
18411  8b17
18412  8b17
18413  8b17  ===> FAC mit 10 multiplizieren <===
18414  8b17
18415  8b17  20 38 8c   mul10  jsr movaf        FAC runden, FAC nach ARG
18416  8b1a  aa               tax
18417  8b1b  f0 10            beq mul10r
18418  8b1d
18419  8b1d  18               clc
18420  8b1e  69 02            adc #$02
```

```
18421  8b20  b0 f2            bcs goover       Behandlung »?overflow error«
18422  8b22
18423  8b22
18424  8b22  ===> (FAC plus ARG) mal zwei <===
18425  8b22
18426  8b22  a2 00      finml6 ldx #$00
18427  8b24  86 70            stx arisgn       Vorzeichen Ergebnis Operation
18428  8b26
18429  8b26  20 55 88         jsr faddc        FAC plus ARG
18430  8b29  e6 63            inc facexp       FAC: EX
18431  8b2b  f0 e7            beq goover       Behandlung »?overflow error«
18432  8b2d
18433  8b2d  60        mul10r rts
18434  8b2e
18435  8b2e
18436  8b2e  ===> GK-Konstante 10 <===
18437  8b2e
18438  8b2e  84 20 00   tenc   .byte $84, $20, $00, $00, $00 18438  8b31  00 00
18439  8b33
18440  8b33
18441  8b33  ===> Behandlung »?division by zero error« <===
18442  8b33
18443  8b33  a2 14      doverr ldx #_erdv0
18444  8b35  4c 3c 4d         jmp error        ind. Behandlung Fehlermeldung
18445  8b38
18446  8b38
18447  8b38  ===> FAC durch 10 dividieren <===
18448  8b38
18449  8b38  20 38 8c   div10  jsr movaf       FAC runden, FAC nach ARG
18450  8b3b
18451  8b3b  a9 2e            lda #<tenc       GK-Konstante 10
18452  8b3d  a0 8b            ldy #>tenc       GK-Konstante 10
18453  8b3f
18454  8b3f  a2 00            ldx #$00
18455  8b41
18456  8b41
18457  8b41  ===> FAC durch VAR ac/yr divid. <===
18458  8b41
18459  8b41  86 70      fdivf  stx arisgn       Vorzeichen Ergebnis Operation
18460  8b43
18461  8b43  20 d4 8b         jsr movfm        RAM Var. ac/yr nach FAC
18462  8b46  4c 4c 8b         jmp fdivt        BASIC-Funktion DIVISION / ($ad)
18463  8b49
18464  8b49
18465  8b49  ===> RAM Var. ac/yr durch FAC dividieren <===
18466  8b49
18467  8b49  20 b4 8a   fdiv   jsr conupk       RAM Var. ac/yr nach ARG
18468  8b4c
18469  8b4c
18470  8b4c  ===> BASIC-Funktion DIVISION / ($ad) <===
18471  8b4c
18472  8b4c  f0 e5      fdivt  beq doverr       Behandlung »?division by zero error«
18473  8b4e
18474  8b4e  20 47 8c         jsr round        FAC runden
18475  8b51
18476  8b51  a9 00            lda #$00
18477  8b53  38               sec
18478  8b54  e5 63            sbc facexp       FAC: EX
```

```
18479  8b56  85 63           sta facexp       FAC: E
18480  8b58
18481  8b58  20 ec 8a        jsr muldiv       ARG EX plus FAC EX
18482  8b5b  e6 63           inc facexp       FAC: EX
18483  8b5d  f0 b5           beq goover       Behandlung »?overflow error«
18484  8b5f
18485  8b5f  a2 fc           ldx #256-_m1     Zeiger auf M1 (4 Bytes)
18486  8b61  a9 01           lda #%00000001   Zähler für 8 Bit
18487  8b63
18488  8b63
18489  8b63  ===> ARG durch ac(FAC)nach ARG <===
18490  8b63
18491  8b63  a4 6b     divide ldy argho       ARG: M4
18492  8b65  c4 64            cpy facho       FAC: M4
18493  8b67  d0 10            bne savquo
18494  8b69
18495  8b69  a4 6c            ldy argmoh      ARG: M3
18496  8b6b  c4 65            cpy facmoh      FAC: M3
18497  8b6d  d0 0a            bne savquo
18498  8b6f
18499  8b6f  a4 6d            ldy argmo       ARG: M2
18500  8b71  c4 66            cpy facmo       FAC: M2
18501  8b73  d0 04            bne savquo
18502  8b75
18503  8b75  a4 6e            ldy arglo       ARG: M1
18504  8b77  c4 67            cpy faclo       FAC: M1
18505  8b79  08        savquo php
18506  8b7a
18507  8b7a  2a               rol a
18508  8b7b  90 09            bcc qshft
18509  8b7d
18510  8b7d  e8               inx             Zeiger in RES
18511  8b7e  95 2b            sta reslo,x     RES: M1
18512  8b80  f0 32            beq ld100
18513  8b82  10 34            bpl divnrm
18514  8b84
18515  8b84  a9 01            lda #%00000001  Zähler für 8 Bit
18516  8b86  28        qshft  plp
18517  8b87  b0 0e            bcs divsub
18518  8b89
18519  8b89
18520  8b89  ===> ARG 1 Bit nach rechts <===
18521  8b89
18522  8b89  06 6e     shfarg asl arglo       ARG: M1
18523  8b8b  26 6d            rol argmo       ARG: M2
18524  8b8d  26 6c            rol argmoh      ARG: M3
18525  8b8f  26 6b            rol argho       ARG: M4
18526  8b91
18527  8b91  b0 e6            bcs savquo
18528  8b93  30 ce            bmi divide      ARG durch ac(FAC)nach ARG
18529  8b95  10 e2            bpl savquo      immer
18530  8b97
18531  8b97
18532  8b97  a8        divsub tay
18533  8b98
18534  8b98  a5 6e            lda arglo       ARG: M1
18535  8b9a  e5 67            sbc faclo       FAC: M1
18536  8b9c  85 6e            sta arglo       ARG: M1
```

```
18537  8b9e
18538  8b9e  a5 6d           lda argmo        ARG: M
18539  8ba0  e5 66           sbc facmo        FAC: M2
18540  8ba2  85 6d           sta argmo        ARG: M2
18541  8ba4
18542  8ba4  a5 6c           lda argmoh       ARG: M3
18543  8ba6  e5 65           sbc facmoh       FAC: M3
18544  8ba8  85 6c           sta argmoh       ARG: M3
18545  8baa
18546  8baa  a5 6b           lda argho        ARG: M4
18547  8bac  e5 64           sbc facho        FAC: M4
18548  8bae  85 6b           sta argho        ARG: M4
18549  8bb0
18550  8bb0  98              tya
18551  8bb1  4c 89 8b        jmp shfarg       ARG 1 Bit nach rechts
18552  8bb4
18553  8bb4
18554  8bb4  a9 40     ld100 lda #$40
18555  8bb6  d0 ce           bne qshft
18556  8bb8
18557  8bb8  0a        divnrm asl a
18558  8bb9  0a              asl a
18559  8bba  0a              asl a
18560  8bbb  0a              asl a
18561  8bbc  0a              asl a
18562  8bbd  0a              asl a
18563  8bbe  85 71           sta facov        Rundungsbyte
18564  8bc0
18565  8bc0  28              plp
18566  8bc1
18567  8bc1
18568  8bc1  ===> RES nach FAC, normalisieren <===
18569  8bc1
18570  8bc1  a5 28     movfr lda resho        RES: M4
18571  8bc3  85 64           sta facho        FAC: M4
18572  8bc5
18573  8bc5  a5 29           lda resmoh       RES: M3
18574  8bc7  85 65           sta facmoh       FAC: M3
18575  8bc9
18576  8bc9  a5 2a           lda resmo        RES: M2
18577  8bcb  85 66           sta facmo        FAC: M2
18578  8bcd
18579  8bcd  a5 2b           lda reslo        RES: M1
18580  8bcf  85 67           sta faclo        FAC: M1
18581  8bd1
18582  8bd1  4c b6 88        jmp normal       FAC normalisieren
18583  8bd4
18584  8bd4
18585  8bd4  ===> RAM Var. ac/yr nach FAC <===
18586  8bd4
18587  8bd4  85 24     movfm sta index1
18588  8bd6  84 25           sty index1+1
18589  8bd8
18590  8bd8  a0 04           ldy #_m1         Zeiger auf M1
18591  8bda  b1 24           lda (index1),y   M1
18592  8bdc  85 67           sta faclo        FAC: M1
18593  8bde
18594  8bde  88              dey
```

```
18595  8bdf  b1 24           lda (index1),y   M2
18596  8be1  85 66           sta facmo        FAC: M2
18597  8be3
18598  8be3  88              dey
18599  8be4  b1 24           lda (index1),y   M3
18600  8be6  85 65           sta facmoh       FAC: M3
18601  8be8
18602  8be8  88              dey
18603  8be9  b1 24           lda (index1),y   M4
18604  8beb  85 68           sta facsgn       FAC: SG
18605  8bed
18606  8bed  09 80           ora #%10000000
18607  8bef  85 64           sta facho        FAC: M4
18608  8bf1
18609  8bf1  88              dey
18610  8bf2  b1 24           lda (index1),y   EX
18611  8bf4  85 63           sta facexp       FAC: EX
18612  8bf6
18613  8bf6  84 71           sty facov        Rundungsbyte
18614  8bf8
18615  8bf8  60              rts
18616  8bf9
18617  8bf9
18618  8bf9  ===> FAC runden, nach TMP FAC2 <===
18619  8bf9
18620  8bf9  a2 5e     mov2f ldx #<tempf2     TMP FAC2
18621  8bfb  2c              .byte _skip2
18622  8bfc
18623  8bfc
18624  8bfc  ===> FAC runden, nach TMP FAC1 <===
18625  8bfc
18626  8bfc  a2 59     mov1f ldx #<tempf1     TMP FAC1
18627  8bfe  a0 00           ldy #>tempf1     TMP FAC1
18628  8c00
18629  8c00
18630  8c00  ===> FAC runden, nach RAM xr/yr <===
18631  8c00
18632  8c00  20 47 8c  movmf  jsr round       FAC runden
18633  8c03
18634  8c03  86 24           stx index1
18635  8c05  84 25           sty index1+1
18636  8c07
18637  8c07  a0 04           ldy #_m1
18638  8c09  a5 67           lda faclo        FAC: M1
18639  8c0b  91 24           sta (index1),y   M1
18640  8c0d
18641  8c0d  88              dey
18642  8c0e  a5 66           lda facmo        FAC: M2
18643  8c10  91 24           sta (index1),y   M2
18644  8c12
18645  8c12  88              dey
18646  8c13  a5 65           lda facmoh       FAC: M3
18647  8c15  91 24           sta (index1),y   M3
18648  8c17
18649  8c17  88              dey
18650  8c18  a5 68           lda facsgn       FAC: SG
18651  8c1a  09 7f           ora #%01111111
18652  8c1c  25 64           and facho        FAC: M4
```

```
18653  8c1e  91 24            sta (index1),y   M4
18654  8c20
18655  8c20  88               dey
18656  8c21  a5 63            lda facexp       FAC: E
18657  8c23  91 24            sta (index1),y   EX
18658  8c25
18659  8c25  84 71            sty facov        0
18660  8c27
18661  8c27  60               rts
18662  8c28
18663  8c28
18664  8c28  ===> ARG nach FAC <===
18665  8c28
18666  8c28  a5 6f      movfa  lda argsgn       ARG: SG
18667  8c2a
18668  8c2a
18669  8c2a  ===> ac nach facexp, Rest ARG nach FAC <===
18670  8c2a
18671  8c2a  85 68      movfa1 sta facsgn       FAC: SG
18672  8c2c
18673  8c2c  a2 05            ldx #_m1+1
18674  8c2e  b5 69      movfal lda argexp-1,x   ARG
18675  8c30  95 62            sta facexp-1,x   FAC
18676  8c32
18677  8c32  ca               dex
18678  8c33  d0 f9            bne movfal
18679  8c35
18680  8c35  86 71            stx facov        Rundungsbyte
18681  8c37
18682  8c37  60               rts
18683  8c38
18684  8c38
18685  8c38  ===> FAC runden, FAC nach ARG <===
18686  8c38
18687  8c38  20 47 8c   movaf  jsr round       FAC runden
18688  8c3b
18689  8c3b
18690  8c3b  ===> FAC nach ARG <===
18691  8c3b
18692  8c3b  a2 06      movef  ldx #_sg+1
18693  8c3d  b5 62      movafl lda facexp-1,x   FAC
18694  8c3f  95 69            sta argexp-1,x   ARG
18695  8c41
18696  8c41  ca               dex
18697  8c42  d0 f9            bne movafl
18698  8c44
18699  8c44  86 71            stx facov        Rundungsbyte
18700  8c46
18701  8c46  60         movrts rts
18702  8c47
18703  8c47
18704  8c47  ===> FAC runden <===
18705  8c47
18706  8c47  a5 63      round  lda facexp       FAC: EX
18707  8c49  f0 fb            beq movrts       FAC = 0
18708  8c4b
18709  8c4b  06 71            asl facov        Rundungsbyte
18710  8c4d  90 f7            bcc movrts
```

```
18711  8c4f
18712  8c4f
18713  8c4f
18713  8c4f  ===> Mantisse von FAC um 1 erhöhen <===
18714  8c4f
18715  8c4f  20 4e 89   incrnd jsr incfac       Mantisse FAC inc. (aufrunden)
18716  8c52  d0 f2             bne movrts
18717  8c54
18718  8c54  4c 17 89          jmp rndshf       Überlaufstelle einschieben
18719  8c57
18720  8c57
18721  8c57  ===> Flag für FAC NULL, PLUS, MINUS nach ac <===
18722  8c57
18723  8c57  a5 63      sign   lda facexp       FAC: EX
18724  8c59  f0 09             beq signrt
18725  8c5b
18726  8c5b
18727  8c5b  ===> Flag für FAC SG PLUS, MINUS nach ac <===
18728  8c5b
18729  8c5b  a5 68      fcsign lda facsgn       FAC: SG
18730  8c5d
18731  8c5d
18732  8c5d  ===> Flag für ac PLUS, MINUS nach ac <===
18733  8c5d
18734  8c5d  2a         fcomps rol a
18735  8c5e  a9 ff             lda #$ff
18736  8c60  b0 02             bcs signrt
18737  8c62
18738  8c62  a9 01             lda #$01
18739  8c64  60         signrt rts
18740  8c65
18741  8c65
18742  8c65  ===> BASIC-Funktion SGN ($b4) <===
18743  8c65
18744  8c65  20 57 8c   sgn    jsr sign         Flag für FAC NULL, PLUS, MINUS nach
                                                ac
18745  8c68
18746  8c68
18747  8c68  ===> 8 Bit INT ac nach FAC <===
18748  8c68
18749  8c68  85 64      float  sta facho        FAC: M4
18750  8c6a
18751  8c6a  a9 00             lda #$00
18752  8c6c  85 65             sta facmoh       FAC: M3
18753  8c6e
18754  8c6e  a2 88             ldx #_expcn+7
18755  8c70
18756  8c70
18757  8c70  ===> 8/16 Bit EX nach FAC, normal. <===
18758  8c70
18759  8c70  a5 64      floats lda facho        FAC: M4
18760  8c72  49 ff             eor #$ff
18761  8c74  2a                 rol a
18762  8c75
18763  8c75
18764  8c75  ===> EX nach FAC, Nullen auffüllen, normal. <===
18765  8c75
18766  8c75  a9 00      floatc lda #$00
```

```
18767  8c77  85 67           sta faclo        FAC: M1
18768  8c79  85 66           sta facmo        FAC: M2
18769  8c7b
18770  8c7b
18771  8c7b  ===> xr nach EX, ac nach SG, FAC normal. <===
18772  8c7b
18773  8c7b  86 63     floatb stx facexp      FAC: EX
18774  8c7d
18775  8c7d  85 71           sta facov        Rundungsbyte
18776  8c7f  85 68           sta facsgn       FAC: SG
18777  8c81
18778  8c81  4c b1 88        jmp fadflt       Test/SG invert., FAC normalisieren
18779  8c84
18780  8c84
18781  8c84  ===> BASIC-Funktion ABS ($b6) <===
18782  8c84
18783  8c84  46 68     abs   lsr facsgn       FAC: SG
18784  8c86  60              rts
18785  8c87
18786  8c87
18787  8c87  ===> FAC mit Var. ac/yr vergleichen <===
18788  8c87
18789  8c87  85 26     fcomp sta index2
18790  8c89  84 27           sty index2+1
18791  8c8b
18792  8c8b  a0 00           ldy #0
18793  8c8d  b1 26           lda (index2),y   EX
18794  8c8f  c8              iny
18795  8c90
18796  8c90  aa              tax              EX
18797  8c91  f0 c4           beq sign         Flag für FAC NULL, PLUS, MINUS nach
                                              ac
18798  8c93
18799  8c93  b1 26           lda (index2),y   M4
18800  8c95  45 68           eor facsgn       FAC: SG
18801  8c97  30 c2           bmi fcsign       SG verschieden
18802  8c99
18803  8c99  e4 63           cpx facexp       FAC: EX
18804  8c9b  d0 21           bne fcompc       Flag für GRÖSSER, KLEINER nach ac
18805  8c9d
18806  8c9d  b1 26           lda (index2),y   M4
18807  8c9f  09 80           ora #%10000000
18808  8ca1
18809  8ca1  c5 64           cmp facho        FAC: M4
18810  8ca3  d0 19           bne fcompc       Flag für GRÖSSER, KLEINER nach ac
18811  8ca5
18812  8ca5  c8              iny
18813  8ca6  b1 26           lda (index2),y   M3
18814  8ca8
18815  8ca8  c5 65           cmp facmoh       FAC: M3
18816  8caa  d0 12           bne fcompc       Flag für GRÖSSER, KLEINER nach ac
18817  8cac
18818  8cac  c8              iny
18819  8cad  b1 26           lda (index2),y   M2
18820  8caf
18821  8caf  c5 66           cmp facmo        FAC: M2
18822  8cb1  d0 0b           bne fcompc       Flag für GRÖSSER, KLEINER nach ac
18823  8cb3
```

```
18824  8cb3  c8              iny
18825  8cb4
18826  8cb4  a9 7f           lda #$7f
18827  8cb6  c5 71           cmp facov        CLC/SEC
18828  8cb8
18829  8cb8  b1 26           lda (index2),y
18830  8cba  e5 67           sbc faclo        FAC: M1
18831  8cbc  f0 2a           beq qintrt       GK-Zahlen gleich
18832  8cbe
18833  8cbe
18834  8cbe  ===> Flag für GRÖSSER, KLEINER nach ac  <===
18835  8cbe18836  8cbe  a5 68      fcompc lda facsgn       FAC: SG
18837  8cc0  90 02           bcc fcompd
18838  8cc2
18839  8cc2  49 ff           eor #$ff
18840  8cc4  4c 5d 8c   fcompd jmp fcomps       Flag für ac PLUS, MINUS nach ac
18841  8cc7
18842  8cc7
18843  8cc7  ===> FAC GK nach INT  <===
18844  8cc7
18845  8cc7  a5 63      qint   lda facexp       FAC: EX
18846  8cc9  f0 4d           beq clrfac       Gleitkomma-Accu löschen
18847  8ccb
18848  8ccb  38              sec
18849  8ccc  e9 a0           sbc #160
18850  8cce  24 68           bit facsgn       FAC: SG
18851  8cd0  10 0a           bpl qishft
18852  8cd2
18853  8cd2  aa              tax
18854  8cd3  a9 ff           lda #$ff
18855  8cd5  8d df 03        sta bits         FAC Overflow Bit
18856  8cd8
18857  8cd8  20 2c 89        jsr negfch       FAC invertieren
18858  8cdb  8a              txa
18859  8cdc
18860  8cdc  a2 63      qishft ldx #<facexp    FAC: EX
18861  8cde
18862  8cde  c9 f9           cmp #$f9
18863  8ce0  10 07           bpl qint1
18864  8ce2
18865  8ce2  20 79 89        jsr shiftr       FAC rechts schieben bis EXP = 0
18866  8ce5  8c df 03        sty bits         FAC Overflow Bit
18867  8ce8
18868  8ce8  60         qintrt rts
18869  8ce9
18870  8ce9
18871  8ce9  a8         qint1  tay
18872  8cea  a5 68           lda facsgn       FAC: SG
18873  8cec  29 80           and #$80
18874  8cee  46 64           lsr facho        FAC: M4
18875  8cf0  05 64           ora facho        FAC: M4
18876  8cf2  85 64           sta facho        FAC: M4
18877  8cf4
18878  8cf4  20 90 89        jsr rolshf       FAC bitweise nach rechts schieben
18879  8cf7  8c df 03        sty bits         FAC Overflow Bit
18880  8cfa
18881  8cfa  60              rts
18882  8cfb
```

```
18883  8cfb
18884  8cfb
18884  8cfb  ===> BASIC-Funktion INT ($b5) <===
18885  8cfb
18886  8cfb  a5 63      int    lda facexp       FAC: EX
18887  8cfd  c9 a0             cmp #160
18888  8cff  b0 20             bcs intrts
18889  8d01
18890  8d01  20 68 aa          jsr rndqnt       GK in FAC runden, nach INT
18891  8d04  84 71             sty facov        Rundungsbyte
18892  8d06
18893  8d06  a5 68             lda facsgn       FAC: SG
18894  8d08  84 68             sty facsgn       FAC: SG
18895  8d0a
18896  8d0a  49 80             eor #$80
18897  8d0c  2a                rol a
18898  8d0d  a9 a0             lda #$a0
18899  8d0f  85 63             sta facexp       FAC: EX
18900  8d11
18901  8d11  a5 67             lda faclo        FAC: M1
18902  8d13  85 09             sta integr       M1 (für Routine fpwrt $8fc1)
18903  8d15
18904  8d15  4c b1 88          jmp fadflt       Test/SG invert., FAC normalisieren
18905  8d18
18906  8d18
18907  8d18  ===> Gleitkomma-Accu löschen <===
18908  8d18
18909  8d18  85 64      clrfac sta facho        FAC: M4
18910  8d1a  85 65             sta facmoh       FAC: M3
18911  8d1c  85 66             sta facmo        FAC: M2
18912  8d1e  85 67             sta faclo        FAC: M1
18913  8d20
18914  8d20  a8                tay
18915  8d21  60         intrts rts
18916  8d22
18917  8d22
18918  8d22                    .end
18919  8d22                    .lib basic.math_4
18920  8d22  put"@basic.math_4"
```

```
18922  8d22
18923  8d22
18924  8d22  ===> Zahlenstr. (Bnk.=xr) nach FAC <===
18925  8d22
18926  8d22  8e da 03   fin    stx finbnk       Bank chrget aus String
18927  8d25
18928  8d25  ===> TMP FAC2 und FAC löschen <===
18929  8d25
18930  8d25  a0 00            ldy #0
18931  8d27  a2 0a            ldx #_sg*2
18932  8d29  94 5f      finzlp sty tempf2+1,x   TMP FAC2
18933  8d2b
18934  8d2b  ca               dex
18935  8d2c  10 fb            bpl finzlp
18936  8d2e
18937  8d2e  90 0f            bcc findgq       (chrget) Ziffer
18938  8d30
18939  8d30  c9 2d            cmp #'-'
18940  8d32  d0 04            bne qplus
18941  8d34
18942  8d34  86 69            stx sgnflg       $ff
18943  8d36  f0 04            beq finc         immer
18944  8d38
18945  8d38
18946  8d38  c9 2b      qplus cmp #'+'
18947  8d3a  d0 05            bne fin1
18948  8d3c
18949  8d3c
18950  8d3c  ===> Dezimalstellen einlesen <===
18951  8d3c
18952  8d3c  20 f5 8d   finc   jsr fnchgt       chrget aus PGM/VAR RAM
18953  8d3f  90 5b      findgq bcc findig       Ziffer
18954  8d41
18955  8d41  c9 2e      fin1   cmp #'.'
18956  8d43  f0 2e            beq findp        Dezimalpunkt bearbeiten
18957  8d45
18958  8d45  c9 45            cmp #'e'
18959  8d47  d0 30            bne fine
18960  8d49
18961  8d49  ===> Exponent einlesen <===
18962  8d49
18963  8d49  20 f5 8d         jsr fnchgt       chrget aus PGM/VAR RAM
18964  8d4c  90 17            bcc fnedg1       Ziffer
18965  8d4e
18966  8d4e  c9 ab            cmp #_minut      MINUS Token
18967  8d50  f0 0e            beq finec1       neg. Exponent
18968  8d52
18969  8d52  c9 2d            cmp #'-'
18970  8d54  f0 0a            beq finec1       neg. Exponent
18971  8d56
18972  8d56  c9 aa            cmp #_plust      PLUS Token
18973  8d58  f0 08            beq finec        pos. Exponent
18974  8d5a
18975  8d5a  c9 2b            cmp #'+'
18976  8d5c  f0 04            beq finec        pos. Exponent
18977  8d5e  d0 07            bne finec2       immer
18978  8d60
18979  8d60
```

```
18980  8d60
18980  8d60  ===> neg. Exponent <===
18981  8d60
18982  8d60  66 62       finec1 ror expsgn
18983  8d62
18984  8d62
18985  8d62  ===> pos. Exponent <===
18986  8d62
18987  8d62  20 f5 8d    finec  jsr fnchgt       chrget aus PGM/VAR RAM
18988  8d65  90 5c       fnedg1 bcc finedg       Ziffer
18989  8d67
18990  8d67  24 62       finec2 bit expsgn
18991  8d69  10 0e              bpl fine
18992  8d6b
18993  8d6b  a9 00              lda #$00
18994  8d6d  38                 sec
18995  8d6e  e5 60              sbc tenexp
18996  8d70  4c 7b 8d           jmp fine1
18997  8d73
18998  8d73
18999  8d73  ===> Dezimalpunkt bearbeiten <===
19000  8d73
19001  8d73  66 61       findp  ror dptflg
19002  8d75  24 61              bit dptflg
19003  8d77  50 c3              bvc finc         Dezimalstellen einlesen
19004  8d79
19005  8d79  a5 60       fine   lda tenexp
19006  8d7b  38         fine1  sec
19007  8d7c  e5 5f              sbc deccnt
19008  8d7e  85 60              sta tenexp
19009  8d80
19010  8d80  f0 12              beq finqng
19011  8d82  10 09              bpl finmul
19012  8d84
19013  8d84  20 38 8b    findiv jsr div10        FAC durch 10 dividieren
19014  8d87  e6 60              inc tenexp
19015  8d89  d0 f9              bne findiv
19016  8d8b  f0 07              beq finqng       immer
19017  8d8d
19018  8d8d
19019  8d8d  20 17 8b    finmul jsr mul10        FAC mit 10 multiplizieren
19020  8d90  c6 60              dec tenexp
19021  8d92  d0 f9              bne finmul
19022  8d94
19023  8d94  a5 69       finqng lda sgnflg
19024  8d96  30 01              bmi negxqs       Sprung NEGATION, Vorzeichen neg.
19025  8d98
19026  8d98  60                 rts
19027  8d99
19028  8d99
19029  8d99  4c fa 8f    negxqs jmp negop        Sprung NEGATION, Vorzeichen neg.
19030  8d9c
19031  8d9c
19032  8d9c  ===> Dezimalstellen nach FAC <===
19033  8d9c
19034  8d9c  48         findig pha              Mantissen-Ziffer
19035  8d9d
19036  8d9d  24 61              bit dptflg
```

```
19037  8d9f  10 02           bpl findg1
19038  8da1
19039  8da1  e6 5f           inc deccnt
19040  8da3  20 17 8b  findg1 jsr mul10      FAC mit 10 multiplizieren
19041  8da6
19042  8da6  68              pla             Mantissen-Ziffer
19043  8da7  38              sec
19044  8da8  e9 30           sbc #'0'
19045  8daa  20 b0 8d        jsr finlog      FAC plus INT in ac
19046  8dad
19047  8dad  4c 3c 8d        jmp finc        Dezimalstellen einlesen
19048  8db0
19049  8db0
19050  8db0  ===> FAC plus INT in ac <===
19051  8db0
19052  8db0  48         finlog pha
19053  8db1
19054  8db1  20 38 8c        jsr movaf       FAC runden, FAC nach ARG
19055  8db4
19056  8db4  68              pla
19057  8db5  20 68 8c        jsr float       8 Bit INT ac nach FAC
19058  8db8
19059  8db8  a5 6f           lda argsgn      ARG: SG
19060  8dba  45 68           eor facsgn      FAC: SG
19061  8dbc  85 70           sta arisgn      Vorzeichen Ergebnis Operation
19062  8dbe
19063  8dbe  a6 63           ldx facexp      FAC: EX
19064  8dc0  4c 48 88        jmp faddt       ARG plus FAC
19065  8dc3
19066  8dc3
19067  8dc3  ===> Exponentenziffern bearbeiten <===
19068  8dc3
19069  8dc3  a5 60      finedg lda tenexp
19070  8dc5  c9 0a           cmp #10
19071  8dc7  90 09           bcc mlex10      Exponenten mit 10 mult.
19072  8dc9
19073  8dc9  a9 64           lda #100
19074  8dcb  24 62           bit expsgn
19075  8dcd  30 21           bmi mlexmi
19076  8dcf
19077  8dcf
19078  8dcf  ===> Behandlung »?overflow error« <===
19079  8dcf
19080  8dcf  4c 5d 89        jmp overr       Behandlung »?overflow error«
19081  8dd2
19082  8dd2
19083  8dd2  ===> Exponenten mit 10 mult. <===
19084  8dd2
19085  8dd2  0a         mlex10 asl a
19086  8dd3  0a                asl a
19087  8dd4  18                clc
19088  8dd5  65 60             adc tenexp
19089  8dd7  0a                asl a
19090  8dd8
19091  8dd8  18                clc
19092  8dd9  a0 00             ldy #0          Zeiger auf EX
19093  8ddb
19094  8ddb  85 79             sta syntmp      temp. tenexp
```

```
19095  8ddd
19096  8ddd  ad da 03          lda finbnk      Bank chrget aus String
19097  8de0  d0 06             bne mx10_1      VAR RAM
19098  8de2
19099  8de2  20 c9 03          jsr indtxt      EX
19100  8de5  4c eb 8d          jmp mx10_2
19101  8de8
19102  8de8
19103  8de8  20 b7 03   mx10_1 jsr ind1_1      EX
19104  8deb
19105  8deb  65 79      mx10_2 adc syntmp      temp. tenexp
19106  8ded  38                sec
19107  8dee  e9 30             sbc #'0'
19108  8df0  85 60      mlexmi sta tenexp
19109  8df2
19110  8df2  4c 62 8d          jmp finec       pos. Exponent
19111  8df5
19112  8df5
19113  8df5  ===> chrget aus PGM/VAR RAM <===
19114  8df5
19115  8df5  ad da 03   fnchgt lda finbnk      Bank chrget aus String
19116  8df8  d0 03             bne fnchg1      VAR RAM
19117  8dfa
19118  8dfa  4c 80 03          jmp chrget      nächstes Zeichen nach ac
19119  8dfd
19120  8dfd
19121  8dfd  ===> chrget aus VAR RAM <===
19122  8dfd
19123  8dfd  e6 24      fnchg1 inc index1
19124  8dff  d0 02             bne fnchg2
19125  8e01  e6 25             inc index1+1
19126  8e03
19127  8e03  a0 00      fnchg2 ldy #0
19128  8e05  20 b7 03          jsr ind1_1      LDA (index1),y aus VAR-RAM
19129  8e08  c9 3a             cmp #':'
19130  8e0a  b0 0a             bcs fnchg3
19131  8e0c
19132  8e0c  c9 20             cmp #_spc
19133  8e0e  f0 ed             beq fnchg1      chrget aus VAR RAM
19134  8e10
19135  8e10  38                sec
19136  8e11  e9 30             sbc #'0'
19137  8e13  38                sec
19138  8e14  e9 d0             sbc #$d0
19139  8e16  60         fnchg3 rts
19140  8e17
19141  8e17
19142  8e17  ===> GK-Konstante 99 999 999.9 <===
19143  8e17
19144  8e17  9b 3e bc   n0999  .byte $9b, $3e, $bc, $1f, $fd
19144  8e1a  1f fd
19145  8e1c
19146  8e1c
19147  8e1c  ===> GK-Konstante 999 999 999 <===
19148  8e1c
19149  8e1c  9e 6e 6b   n9999  .byte $9e, $6e, $6b, $27, $fd
19149  8e1f  27 fd
19150  8e21
```

```
19151  8e21
19152  8e21
19152  8e21  ===> GK-Konstante 1 000 000 000 <===
19153  8e21
19154  8e21  9e 6e 6b   nmil   .byte $9e, $6e, $6b, $28, $00
19154  8e24  28 00
19155  8e26
19156  8e26
19157  8e26  ===> Ausgabe »in .. Zeilennummer« <===
19158  8e26
19159  8e26  20 81 92   inprt  jsr cprimm       folg. Text ausg./überspringen
19160  8e29  20 49 4e          .byte ' in ', $00
19160  8e2c  20 00
19161  8e2e
19162  8e2e
19163  8e2e  ===> Ausgabe akt. Zeilennummer <===
19164  8e2e
19165  8e2e  a5 3c      curprt lda curlin+1     High-Byte akt. Zeilennummer
19166  8e30  a6 3b             ldx curlin      Low-Byte akt. Zeilennummer
19167  8e32
19168  8e32
19169  8e32  ===> xr/ac von hex nach dez, Ausgabe in ASCII <===
19170  8e32
19171  8e32  85 64      linprt sta facho       FAC: M4
19172  8e34  86 65             stx facmoh      FAC: M3
19173  8e36
19174  8e36  a2 90             ldx #_expcn+15
19175  8e38  38                sec
19176  8e39  20 75 8c          jsr floatc      EX nach FAC, Nullen auffüllen,
                                               normal.
19177  8e3c  20 44 8e          jsr foutc       FAC nach String, Adr. nach ac/yr
19178  8e3f  4c e2 55          jmp strout      Ausgabe String ac/yr
19179  8e42
19180  8e42
19181  8e42  ===> FAC nach String, Adr. nach ac/yr <===
19182  8e42
19183  8e42  a0 01      fout   ldy #1          Zeiger in Stringpuffer
19184  8e44
19185  8e44
19186  8e44  ===> FAC nach String, Adr. nach ac/yr <===
19187  8e44
19188  8e44  a9 20      foutc  lda #_spc
19189  8e46  24 68             bit facsgn      FAC: SG
19190  8e48  10 02             bpl fout1
19191  8e4a
19192  8e4a  a9 2d             lda #'-'
19193  8e4c  99 ff 00   fout1  sta fbuffr-1,y
19194  8e4f
19195  8e4f  85 68             sta facsgn      FAC: SG
19196  8e51  84 72             sty fbufpt      ADDRL Zeiger in Puffer
19197  8e53
19198  8e53  c8                iny
19199  8e54  a9 30             lda #'0'        Null vorbesetzen
19200  8e56
19201  8e56  a6 63             ldx facexp      FAC: EX
19202  8e58  d0 03             bne fout10
19203  8e5a
19204  8e5a
```

```
19204  8e5a  ===> FAC = 0 <===
19205  8e5a
19206  8e5a  4c 69 8f          jmp fout19       ac und EOM ins Stringpuffer,
                                                Abschluß
19207  8e5d
19208  8e5d
19209  8e5d  a9 00       fout10 lda #0           vorbesetzen deccnt
19210  8e5f
19211  8e5f  e0 80              cpx #_expcn-1
19212  8e61  f0 02              beq fout37       FAC zwischen 1/2 und 1
19213  8e63  b0 09              bcs fout7        FAC > 1
19214  8e65
19215  8e65
19216  8e65  ===> FAC mit 1 000 000 000 mult. <===
19217  8e65
19218  8e65  a9 21       fout37 lda #<nmil       GK-Konstante 1 000 000 000
19219  8e67  a0 8e              ldy #>nmil       GK-Konstante 1 000 000 000
19220  8e69  20 08 8a          jsr rommlt       ROM Konst. ac/yr mit FAC mult.
19221  8e6c
19222  8e6c  a9 f7              lda #256-9
19223  8e6e  85 5f       fout7  sta deccnt
19224  8e70
19225  8e70
19226  8e70  ===> FAC mit 999 999 999 vergleichen <===
19227  8e70
19228  8e70  a9 1c       fout4  lda #<n9999      GK-Konstante 999 999 999
19229  8e72  a0 8e              ldy #>n9999      GK-Konstante 999 999 999
19230  8e74  20 87 8c          jsr fcomp        FAC mit Var. ac/yr vergleichen
19231  8e77
19232  8e77  f0 1e              beq bigges       gleich
19233  8e79  10 12              bpl fout9        FAC größer
19234  8e7b
19235  8e7b
19236  8e7b  ===> FAC mit 99 999 999.9 vergleichen <===
19237  8e7b
19238  8e7b  a9 17       fout3  lda #<n0999      GK-Konstante 99 999 999.9
19239  8e7d  a0 8e              ldy #>n0999      GK-Konstante 99 999 999.9
19240  8e7f  20 87 8c          jsr fcomp        FAC mit Var. ac/yr vergleichen
19241  8e82
19242  8e82  f0 02              beq fout38       gleich
19243  8e84  10 0e              bpl fout5        FAC größer
19244  8e86
19245  8e86  20 17 8b   fout38 jsr mul10        FAC mit 10 multiplizieren
19246  8e89  c6 5f              dec deccnt
19247  8e8b  d0 ee              bne fout3        FAC mit 99 999 999.9 vergleichen
19248  8e8d
19249  8e8d  20 38 8b   fout9  jsr div10        FAC durch 10 dividieren
19250  8e90  e6 5f              inc deccnt
19251  8e92  d0 dc              bne fout4        FAC mit 999 999 999 vergleichen
19252  8e94
19253  8e94  20 0e 8a   fout5  jsr faddh        0.5 zu FAC addieren
19254  8e97
19255  8e97  20 c7 8c   bigges jsr qint         FAC GK nach INT
19256  8e9a
19257  8e9a  a2 01              ldx #1
19258  8e9c  a5 5f              lda deccnt
19259  8e9e  18                 clc
19260  8e9f  69 0a              adc #10
```

```
19261  8ea1  30 09           bmi foutpi
19262  8ea3
19263  8ea3  c9 0b           cmp #11
19264  8ea5  b0 06           bcs fout6
19265  8ea7
19266  8ea7  69 ff           adc #255
19267  8ea9  aa              tax
19268  8eaa
19269  8eaa  a9 02           lda #2
19270  8eac  38       foutpi sec
19271  8ead  e9 02    fout6  sbc #2
19272  8eaf  85 60           sta tenexp
19273  8eb1
19274  8eb1  86 5f           stx deccnt
19275  8eb3  8a              txa
19276  8eb4  f0 02           beq fout39      Dezimalpunkt in Stringpuffer
19277  8eb6  10 13           bpl fout8       Ziffern Zahlenstring einzeln
                                             berechnen
19278  8eb8
19279  8eb8
19280  8eb8  ===> Dezimalpunkt in Stringpuffer <===
19281  8eb8
19282  8eb8  a4 72    fout39 ldy fbufpt      ADDRL Zeiger in Puffer
19283  8eba  a9 2e           lda #'.'
19284  8ebc  c8              iny
19285  8ebd  99 ff 00        sta fbuffr-1,y
19286  8ec0
19287  8ec0  8a              txa
19288  8ec1  f0 06           beq fout16
19289  8ec3
19290  8ec3  a9 30           lda #'0'
19291  8ec5  c8              iny
19292  8ec6  99 ff 00        sta fbuffr-1,y
19293  8ec9
19294  8ec9  84 72    fout16 sty fbufpt      ADDRL Zeiger in Puffer
19295  8ecb
19296  8ecb
19297  8ecb  ===> Ziffern Zahlenstring einzeln berechnen <===
19298  8ecb
19299  8ecb  a0 00    fout8  ldy #0
19300  8ecd
19301  8ecd  a2 80    foutim ldx #$80
19302  8ecf
19303  8ecf  a5 67    fout2  lda faclo       FAC: M1
19304  8ed1  18              clc
19305  8ed2  79 7e 8f        adc foutbl+3,y  M4
19306  8ed5  85 67           sta faclo       FAC: M1
19307  8ed7
19308  8ed7  a5 66           lda facmo       FAC: M2
19309  8ed9  79 7d 8f        adc foutbl+2,y  M3
19310  8edc  85 66           sta facmo       FAC: M2
19311  8ede
19312  8ede  a5 65           lda facmoh      FAC: M3
19313  8ee0  79 7c 8f        adc foutbl+1,y  M2
19314  8ee3  85 65           sta facmoh      FAC: M3
19315  8ee5
19316  8ee5  a5 64           lda facho       FAC: M4
19317  8ee7  79 7b 8f        adc foutbl,y    m1
```

```
19318  8eea  85 64            sta facho        FAC: M4
19319  8eec
19320  8eec  e8              inx
19321  8eed
19322  8eed  b0 04           bcs fout41
19323  8eef  10 de           bpl fout2
19324  8ef1  30 02           bmi fout40       immer
19325  8ef3
19326  8ef3
19327  8ef3  30 da     fout41 bmi fout2
19328  8ef5
19329  8ef5  8a        fout40 txa
19330  8ef6  90 04           bcc foutyp
19331  8ef8
19332  8ef8  49 ff           eor #%11111111
19333  8efa  69 0a           adc #10
19334  8efc
19335  8efc  69 2f     foutyp adc #$2f
19336  8efe
19337  8efe  ===> Zeiger auf nächste Konstante <===
19338  8efe
19339  8efe  c8              iny
19340  8eff  c8              iny
19341  8f00  c8              iny
19342  8f01  c8              iny
19343  8f02  84 49           sty varpnt       Zeiger in foutbl
19344  8f04
19345  8f04  a4 72           ldy fbufpt       ADDRL Zeiger in Puffer
19346  8f06  c8              iny
19347  8f07
19348  8f07  aa              tax
19349  8f08  29 7f           and #%01111111
19350  8f0a  99 ff 00        sta fbuffr-1,y
19351  8f0d
19352  8f0d  c6 5f           dec deccnt
19353  8f0f  d0 06           bne stxbuf
19354  8f11
19355  8f11  a9 2e           lda #'.'
19356  8f13  c8              iny
19357  8f14  99 ff 00        sta fbuffr-1,y
19358  8f17
19359  8f17  84 72     stxbuf sty fbufpt       ADDRL Zeiger in Puffer
19360  8f19
19361  8f19  a4 49           ldy varpnt       Zeiger in foutbl
19362  8f1b
19363  8f1b  8a              txa
19364  8f1c  49 ff           eor #%11111111
19365  8f1e  29 80           and #%10000000
19366  8f20  aa              tax
19367  8f21
19368  8f21  c0 24           cpy #fdcend-foutbl
19369  8f23  f0 04           beq fouldy
19370  8f25
19371  8f25  c0 3c           cpy #timend-foutbl
19372  8f27  d0 a6           bne fout2
19373  8f29
19374  8f29  a4 72     fouldy ldy fbufpt       ADDRL Zeiger in Puffer
19375  8f2b
```

```
19376  8f2b  b9 ff 00   fout11 lda fbuffr-1,y
19377  8f2e  88                dey
19378  8f2f
19379  8f2f  c9 30             cmp #'0'
19380  8f31  f0 f8             beq fout11
19381  8f33
19382  8f33  c9 2e             cmp #'.'
19383  8f35  f0 01             beq fout12
19384  8f37
19385  8f37  c8                iny
19386  8f38
19387  8f38  a9 2b      fout12 lda #'+'
19388  8f3a
19389  8f3a  a6 60             ldx tenexp
19390  8f3c  f0 2e             beq fout17       EOM in Stringpuffer, Abschluß
19391  8f3e  10 08             bpl fout14
19392  8f40
19393  8f40  a9 00             lda #0
19394  8f42  38                sec
19395  8f43  e5 60             sbc tenexp
19396  8f45  aa                tax
19397  8f46
19398  8f46  a9 2d             lda #'-'
19399  8f48
19400  8f48  99 01 01   fout14 sta fbuffr+1,y
19401  8f4b
19402  8f4b  a9 45             lda #'e'
19403  8f4d  99 00 01          sta fbuffr,y
19404  8f5019405  8f50  8a                txa
19406  8f51  a2 2f             ldx #$2f
19407  8f53  38                sec
19408  8f54  e8         fout15 inx
19409  8f55  e9 0a             sbc #$0a
19410  8f57  b0 fb             bcs fout15
19411  8f59
19412  8f59  69 3a             adc #$3a
19413  8f5b  99 03 01          sta fbuffr+3,y
19414  8f5e
19415  8f5e  8a                txa
19416  8f5f  99 02 01          sta fbuffr+2,y
19417  8f62
19418  8f62  a9 00             lda #_eom
19419  8f64  99 04 01          sta fbuffr+4,y
19420  8f67  f0 08             beq fout20       immer
19421  8f69
19422  8f69
19423  8f69  ===> ac und EOM ins Stringpuffer, Abschluß <===
19424  8f69
19425  8f69  99 ff 00   fout19 sta fbuffr-1,y
19426  8f6c
19427  8f6c
19428  8f6c  ===> EOM in Stringpuffer, Abschluß <===
19429  8f6c
19430  8f6c  a9 00      fout17 lda #_eom
19431  8f6e  99 00 01          sta fbuffr,y
19432  8f71
19433  8f71  a9 00      fout20 lda #<fbuffr
19434  8f73  a0 01             ldy #>fbuffr
```

```
19435  8f75  60               rts
19436  8f76
19437  8f76
19438  8f76  ===> GK-Konstante 0.5 für SQR <===
19439  8f76
19440  8f76  80 00 00   fhalf  .byte $80, $00, $00, $00, $00
19440  8f79  00 00
19441  8f7b
19442  8f7b
19443  8f7b  ===> Tab. Mantissen GK nach ASCII <===
19444  8f7b
19445  8f7b  fa 0a 1f   foutbl .byte $fa, $0a, $1f, $00
19445  8f7e  00
19446  8f7f
19447  8f7f
19448  8f7f  ===> Mantisse GK nach ASCII: 10 000 000 <===
19449  8f7f
19450  8f7f  00 98 96   foutb1 .byte $00, $98, $96, $80
19450  8f82  80
19451  8f83
19452  8f83
19453  8f83  ===> Mantisse GK nach ASCII: -1 000 000 <===
19454  8f83
19455  8f83  ff f0 bd   foutb2 .byte $ff, $f0, $bd, $c0
19455  8f86  c0
19456  8f87
19457  8f87
19458  8f87  ===> Mantisse GK nach ASCII:    100 000 <===
19459  8f87
19460  8f87  00 01 86   foutb3 .byte $00, $01, $86, $a0
19460  8f8a  a0
19461  8f8b
19462  8f8b
19463  8f8b  ===> Mantisse GK nach ASCII:    -10 000 <===
19464  8f8b
19465  8f8b  ff ff d8   foutb4 .byte $ff, $ff, $d8, $f0
19465  8f8e  f0
19466  8f8f
19467  8f8f
19468  8f8f  ===> Mantisse GK nach ASCII:      1 000 <===
19469  8f8f
19470  8f8f  00 00 03   foutb5 .byte $00, $00, $03, $e8
19470  8f92  e8
19471  8f93
19472  8f93
19473  8f93  ===> Mantisse GK nach ASCII:       -100 <===
19474  8f93
19475  8f93  ff ff ff   foutb6 .byte $ff, $ff, $ff, $9c
19475  8f96  9c
19476  8f97
19477  8f97
19478  8f97  ===> Mantisse GK nach ASCII:         10 <===
19479  8f97
19480  8f97  00 00 00   foutb7 .byte $00, $00, $00, $0a
19480  8f9a  0a
19481  8f9b
19482  8f9b
```

```
19483  8f9b  ===> Mantisse GK nach ASCII:          -1 <===
19484  8f9b
19485  8f9b  ff ff ff   foutb8 .byte $ff, $ff, $ff, $ff
19485  8f9e  ff
19486  8f9f            fdcend
19487  8f9f
19488  8f9f
19489  8f9f  ===> Mantisse TIME nach ASCII: -2 160 000 <===
19490  8f9f
19491  8f9f  ff df 0a   timcon .byte $ff, $df, $0a, $80
19491  8fa2  80
19492  8fa3
19493  8fa3
19494  8fa3  ===> Mantisse TIME nach ASCII:    216 000 <===
19495  8fa3
19496  8fa3  00 03 4b   timco1 .byte $00, $03, $4b, $c0
19496  8fa6  c0
19497  8fa7
19498  8fa7
19499  8fa7  ===> Mantisse TIME nach ASCII:    -36 000 <===
19500  8fa7
19501  8fa7  ff ff 73   timco2 .byte $ff, $ff, $73, $60
19501  8faa  60
19502  8fab
19503  8fab
19504  8fab  ===> Mantisse TIME nach ASCII:      3 600 <===
19505  8fab
19506  8fab  00 00 0e   timco3 .byte $00, $00, $0e, $10
19506  8fae  10
19507  8faf
19508  8faf
19509  8faf  ===> Mantisse TIME nach ASCII:       -600 <===
19510  8faf
19511  8faf  ff ff fd   timco4 .byte $ff, $ff, $fd, $a8
19511  8fb2  a8
19512  8fb3
19513  8fb3
19514  8fb3  ===> Mantisse TIME nach ASCII:         60 <===
19515  8fb3
19516  8fb3  00 00 00   timco5 .byte $00, $00, $00, $3c
19516  8fb6  3c
19517  8fb7            timend
19518  8fb7
19519  8fb7
19520  8fb7  ===> BASIC-Funktion SQR ($ba) <===
19521  8fb7
19522  8fb7  20 38 8c   sqr    jsr movaf        FAC runden, FAC nach ARG
19523  8fba
19524  8fba  a9 76            lda #<fhalf      GK-Konstante 0.5 für SQR
19525  8fbc  a0 8f            ldy #>fhalf      GK-Konstante 0.5 für SQR
19526  8fbe  20 d4 8b   fpwr   jsr movfm        RAM Var. ac/yr nach FAC
19527  8fc1
19528  8fc1
19529  8fc1  ===> BASIC-Funktion POTENZ  ($ae) <===
19530  8fc1
19531  8fc1  f0 70      fpwrt  beq exp          BASIC-Funktion EXP ($bd)
19532  8fc3
19533  8fc3  a5 6a            lda argexp       ARG: EX
```

```
19534  8fc5  d0 03           bne fpwrt1
19535  8fc7
19536  8fc7  4c d8 88        jmp zerof1
19537  8fca
19538  8fca
19539  8fca  a2 50     fpwrt1 ldx #<defpnt     ADDRL Zeiger FN Variable
19540  8fcc  a0 00            ldy #>defpnt     ADDRL Zeiger FN Variable
19541  8fce  20 00 8c         jsr movmf        FAC runden, nach RAM xr/yr
19542  8fd1
19543  8fd1  a5 6f            lda argsgn       ARG: SG
19544  8fd3  10 0f            bpl fpwr1
19545  8fd5
19546  8fd5  20 fb 8c         jsr int          BASIC-Funktion INT ($b5)
19547  8fd8
19548  8fd8  a9 50            lda #<defpnt     ADDRL Zeiger FN Variable
19549  8fda  a0 00            ldy #>defpnt     ADDRL Zeiger FN Variable
19550  8fdc  20 87 8c         jsr fcomp        FAC mit Var. ac/yr vergleichen
19551  8fdf  d0 03            bne fpwr1
19552  8fe1
19553  8fe1  98               tya
19554  8fe2  a4 09            ldy integr       M1 (aus Routine int $8cfb)
19555  8fe4  20 2a 8c  fpwr1  jsr movfa1       ac nach facexp, Rest ARG nach FAC
19556  8fe7
19557  8fe7  98               tya
19558  8fe8  48               pha
19559  8fe9
19560  8fe9  20 ca 89         jsr log          BASIC-Funktion LOG ($bc)
19561  8fec  a9 50            lda #<defpnt     ADDRL Zeiger FN Variable
19562  8fee  a0 00            ldy #>defpnt     ADDRL Zeiger FN Variable
19563  8ff0  20 24 8a         jsr fmult        FAC mal RAM VAR ac/yr
19564  8ff3
19565  8ff3  20 33 90         jsr exp          BASIC-Funktion EXP ($bd)
19566  8ff6
19567  8ff6  68               pla
19568  8ff7  4a               lsr a
19569  8ff8  90 0a            bcc negrts
19570  8ffa
19571  8ffa
19572  8ffa  ===> BASIC-Funktion NEG. VORZEICHEN <===
19573  8ffa
19574  8ffa  a5 63     negop  lda facexp       FAC: EX
19575  8ffc  f0 06            beq negrts
19576  8ffe
19577  8ffe  a5 68            lda facsgn       FAC: SG
19578  9000  49 ff            eor #%11111111
19579  9002  85 68            sta facsgn       FAC: SG
19580  9004
19581  9004  60        negrts rts
19582  9005
19583  9005
19584  9005  ===> Konstante 1.44269504 = 1/log(2) <===
19585  9005
19586  9005  81 38 aa   logeb2 .byte $81, $38, $aa, $3b, $29
19586  9008  3b 29
19587  900a
19588  900a
```

```
19589  900a  ===> Grad (7) für Polynom EXP <===
19590  900a
19591  900a  07          expcon .byte $07
19592  900b
19593  900b
19594  900b
19594  900b  ===> GK-Koeffizient 7: 2.149 876 37 e-5 <===
19595  900b
19596  900b  71 34 58    expco7 .byte $71, $34, $58, $3e, $56
19596  900e  3e 56
19597  9010
19598  9010
19599  9010  ===> GK-Koeffizient 6: 1.435 231 4  e-4 <===
19600  9010
19601  9010  74 16 7e    expco6 .byte $74, $16, $7e, $b3, $1b
19601  9013  b3 1b
19602  9015
19603  9015
19604  9015  ===> GK-Koeffizient 5: 1.342 263 48 e-3 <===
19605  9015
19606  9015  77 2f ee    expco5 .byte $77, $2f, $ee, $e3, $85
19606  9018  e3 85
19607  901a
19608  901a
19609  901a  ===> GK-Koeffizient 4: 9.614 017 01 e-3 <===
19610  901a
19611  901a  7a 1d 84    expco4 .byte $7a, $1d, $84, $1c, $2a
19611  901d  1c 2a
19612  901f
19613  901f
19614  901f  ===> GK-Koeffizient 3: 0.055 505 126 9 <===
19615  901f
19616  901f  7c 63 59    expco3 .byte $7c, $63, $59, $58, $0a
19616  9022  58 0a
19617  9024
19618  9024
19619  9024  ===> GK-Koeffizient 2: 0.240 226 385 <===
19620  9024
19621  9024  7e 75 fd    expco2 .byte $7e, $75, $fd, $e7, $c6
19621  9027  e7 c6
19622  9029
19623  9029
19624  9029  ===> GK-Koeffizient 1: 0.693 147 186 <===
19625  9029
19626  9029  80 31 72    expco1 .byte $80, $31, $72, $18, $10
19626  902c  18 10
19627  902e  81 00 00    expco0 .byte $81, $00, $00, $00, $00
19627  9031  00 00
19628  9033
19629  9033
19630  9033  ===> BASIC-Funktion EXP ($bd) <===
19631  9033
19632  9033  a9 05       exp    lda #<logeb2     Konstante 1.44269504 = 1/log(2)
19633  9035  a0 90              ldy #>logeb2     Konstante 1.44269504 = 1/log(2)
19634  9037  20 08 8a          jsr rommlt       ROM Konst. ac/yr mit FAC mult.
19635  903a
19636  903a  a5 71              lda facov        Rundungsbyte
19637  903c  69 50              adc #80
```

```
19638  903e  90 03           bcc stold
19639  9040
19640  9040  20 4f 8c        jsr incrnd      Mantisse von FAC um 1 erhöhen
19641  9043  85 58     stold  sta jmper+2     ADDRH Funktions Routine
19642  9045
19643  9045  20 3b 8c        jsr movef       FAC nach ARG
19644  9048
19645  9048  a5 63           lda facexp      FAC: EX
19646  904a  c9 88           cmp #_expcn+7
19647  904c  90 03           bcc exp1
19648  904e
19649  904e  20 09 8b  gomldv jsr mldvex
19650  9051  20 fb 8c  exp1   jsr int         BASIC-Funktion INT ($b5)
19651  9054
19652  9054  a5 09           lda integr      M1 (aus Routine int $8cfb)
19653  9056  18              clc
19654  9057  69 81           adc #_expcn
19655  9059  f0 f3           beq gomldv
19656  905b
19657  905b  38              sec
19658  905c  e9 01           sbc #1
19659  905e  48              pha             EX
19660  905f
19661  905f  ===> FAC und ARG tauschen <===
19662  905f
19663  905f  a2 05           ldx #_sg
19664  9061  b5 6a     swaplp lda argexp,x    ARG
19665  9063  b4 63           ldy facexp,x    FAC
19666  9065
19667  9065  95 63           sta facexp,x    FAC
19668  9067  94 6a           sty argexp,x    ARG
19669  9069
19670  9069  ca              dex
19671  906a  10 f5           bpl swaplp
19672  906c
19673  906c  a5 58           lda jmper+2     ADDRH Funktions Routine
19674  906e  85 71           sta facov       Rundungsbyte
19675  9070
19676  9070  20 31 88        jsr fsubt       BASIC-Funktion MINUS ($ab)
19677  9073  20 fa 8f        jsr negop       BASIC-Funktion NEG. VORZEICHEN
19678  9076
19679  9076  a9 0a           lda #<expcon    Grad (7) für Polynom EXP
19680  9078  a0 90           ldy #>expcon    Grad (7) für Polynom EXP
19681  907a  20 9c 90        jsr poly
19682  907d
19683  907d  a9 00           lda #0
19684  907f  85 70           sta arisgn      Vorzeichen Ergebnis Operation
19685  9081
19686  9081  68              pla             EX
19687  9082  20 ee 8a        jsr mldexp      ac EX plus FAC EX
19688  9085  60              rts
19689  9086
19690  9086
19691  9086  ===> Polynom-Auswertung <===
19692  9086
19693  9086  85 72     polyx  sta polypt      ADDRL Zeiger in Polynom
19694  9088  84 73           sty polypt+1    ADDRH Zeiger in Polynom
19695  908a
```

```
19696  908a  20 fc 8b          jsr mov1f        FAC runden, nach TMP FAC1
19697  908d
19698  908d  a9 59             lda #<tempf1     TMP FAC1
19699  908f  20 24 8a          jsr fmult        FAC mal RAM VAR ac/yr
19700  9092  20 a0 90          jsr poly1
19701  9095
19702  9095  a9 59             lda #<tempf1     TMP FAC1
19703  9097  a0 00             ldy #>tempf1     TMP FAC1
19704  9099  4c 24 8a          jmp fmult        FAC mal RAM VAR ac/yr
19705  909c
19706  909c
19707  909c  85 72       poly   sta polypt       ADDRL Zeiger in Polynom
19708  909e  84 73              sty polypt+1     ADDRH Zeiger in Polynom
19709  90a0
19710  90a0  20 f9 8b    poly1  jsr mov2f       FAC runden, nach TMP FAC2
19711  90a3  b1 72              lda (polypt),y   ADDRL Zeiger in Polynom
19712  90a5  85 69              sta sgnflg
19713  90a7
19714  90a7  a4 72              ldy polypt       ADDRL Zeiger in Polynom
19715  90a9  c8                 iny
19716  90aa  98                 tya
19717  90ab  d0 02              bne poly3
19718  90ad  e6 73              inc polypt+1     ADDRH Zeiger in Polynom
19719  90af
19720  90af  85 72       poly3  sta polypt       ADDRL Zeiger in Polynom
19721  90b1
19722  90b1  a4 73              ldy polypt+1     ADDRH Zeiger in Polynom
19723  90b3  20 08 8a    poly2  jsr rommlt       ROM Konst. ac/yr mit FAC mult.
19724  90b6
19725  90b6  ===> Polynom Zeiger auf nächsten Koeffizienten <===
19726  90b6
19727  90b6  a5 72              lda polypt       ADDRL Zeiger in Polynom
19728  90b8  a4 73              ldy polypt+1     ADDRH Zeiger in Polynom
19729  90ba  18                 clc
19730  90bb  69 05              adc #_sg
19731  90bd  90 01              bcc poly4
19732  90bf  c8                 iny
19733  90c0
19734  90c0  85 72       poly4  sta polypt       ADDRL Zeiger in Polynom
19735  90c2  84 73              sty polypt+1     ADDRH Zeiger in Polynom
19736  90c4
19737  90c4  20 12 8a          jsr romadd       ROM Konst. ac/yr zu FAC addieren
19738  90c7
19739  90c7  a9 5e             lda #<tempf2     TMP FAC2
19740  90c9  a0 00             ldy #>tempf2     TMP FAC2
19741  90cb
19742  90cb  c6 69             dec sgnflg
19743  90cd  d0 e4             bne poly219744  90cf
19745  90cf  60                rts
19746  90d0
19747  90d0
19748  90d0                    .end
19749  90d0                    .lib basic.io
19750  90d0  put"@basic.io"
```

```
19752  90d0
19753  90d0
19754  90d0  ===> Behandlung I/O-Fehler <===
19755  90d0
19756  90d0  aa         erexit tax              Fehlernummer
19757  90d1  d0 02             bne erexiy
19758  90d3
19759  90d3
19760  90d3  ===> Behandlung I/O-BREAK <===
19761  90d3
19762  90d3  a2 1e      brkxit ldx #_erbrk
19763  90d5
19764  90d5  ===> ind. Behandlung Fehlermeldung <===
19765  90d5
19766  90d5  4c 3c 4d   erexiy jmp error        ind. Behandlung Fehlermeldung
19767  90d8
19768  90d8
19769  90d8  ===> BASIC-Statement OPEN ($9f) <===
19770  90d8
19771  90d8  20 45 a8   copen  jsr mapio        I/O-Chips einblenden
19772  90db  20 c0 ff          jsr kopen        log. File üb. Vector öffnen, Ausgabe
                                                fn (sprungv.)
19773  90de  60                rts
19774  90df
19775  90df
19776  90df  ===> Ausgabe ac akt. Kanal, Fehlertest (BASIC) <===
19777  90df
19778  90df  20 69 92   outch  jsr cbsout       ind. Ausgabe ac auf akt. Kanal
19779  90e2  b0 ec             bcs erexit       Behandlung I/O-Fehler
19780  90e4  60                rts
19781  90e5
19782  90e5
19783  90e5  ===> Eingabe akt. Kanal nach ac, Fehlertest (BASIC) <===
19784  90e5
19785  90e5  20 63 92   inchr  jsr cbasin       ind. Zeich. akt. Kanal nach ac
19786  90e8  b0 e6             bcs erexit       Behandlung I/O-Fehler
19787  90ea  60                rts
19788  90eb
19789  90eb
19790  90eb  ===> O-Kanal öffnen/listen, Fehlertest (BASIC) <===
19791  90eb
19792  90eb  48         coout  pha
19793  90ec
19794  90ec  20 45 a8           jsr mapio        I/O-Chips einblenden
19795  90ef  20 c9 ff           jsr kckout       Ausgabe-Kanal üb. Vector öffnen
                                                (Sprungv.)
19796  90f2  20 43 92           jsr dschk        für FA > 3 DS$ rücksetzen
19797  90f5  aa                 tax              Fehlernummer
19798  90f6
19799  90f6  68                 pla
19800  90f7  90 03              bcc cckou1
19801  90f9
19802  90f9  8a                 txa              Fehlernummer
19803  90fa  b0 d4              bcs erexit       immer
19804  90fc
19805  90fc
19806  90fc  60         cckou1 rts
19807  90fd
```

```
19808  90fd
19809  90fd  ===> I-Kanal öffnen, Fehlertest (BASIC) <===
19810  90fd
19811  90fd  20 45 a8   coin   jsr mapio        I/O-Chips einblenden
19812  9100  20 c6 ff          jsr kchkin       Eingabekanal üb. Vector öffnen
                                                (Sprungv.)
19813  9103  20 43 92          jsr dschk        für FA > 3 DS$ rücksetzen
19814  9106  b0 c8             bcs erexit       Behandlung I/O-Fehler
19815  9108  60                 rts
19816  9109
19817  9109
19818  9109  ===> Eingabe akt. Kanal nach ac, Fehlertest (BASIC) <===
19819  9109
19820  9109  20 45 a8   cgetl  jsr mapio        I/O-Chips einblenden
19821  910c  20 e4 ff          jsr kgetin       Zeichen üb. Vector vom akt. Kanal
                                                nach ac (Sprungv.)
19822  910f  b0 c2             bcs brkxit       Behandlung I/O-BREAK
19823  9111  60                 rts
19824  9112
19825  9112
19826  9112  ===> BASIC-Statement SAVE ($94) <===
19827  9112
19828  9112  20 ae 91   save   jsr plsv
19829  9115
19830  9115
19831  9115  ===> SAVE, Param. eingelesen <===
19832  9115
19833  9115  ae 10 12   savenp ldx txttop       ADDRL Obergrenze akt. PGM
19834  9118  ac 11 12          ldy txttop+1     ADDRH Obergrenze akt. PGM
19835  911b
19836  911b  a9 2d             lda #<txttab     ADDRL Untergrenze akt. PGM
19837  911d
19838  911d
19839  911d  ===> Teil der save-Routine <===
19840  911d
19841  911d  20 45 a8   savenb jsr mapio        I/O-Chips einblenden
19842  9120  20 d8 ff          jsr ksaves       save ab ac bis xr, yr (Sprungv.)
19843  9123  20 43 92          jsr dschk        für FA > 3 DS$ rücksetzen
19844  9126  b0 a8             bcs erexit       Behandlung I/O-Fehler
19845  9128
19846  9128  60                 rts
19847  9129
19848  9129
19849  9129  ===> BASIC-Statement VERIFY ($95) <===
19850  9129
19851  9129  a9 01      verify lda #1           VERIFY
19852  912b  2c                .byte _skip2
19853  912c
19854  912c
19855  912c  ===> BASIC-Statement LOAD ($93) <===
19856  912c
19857  912c  a9 00      load   lda #0           LOAD
19858  912e  85 0c             sta bverck       Flag LOAD=0, VERIFY=1 (BASIC)
19859  9130
19860  9130  20 ae 91          jsr plsv
19861  9133
19862  9133
19863  9133
```

```
19863  9133  ===> LOAD/VERIFY, Param. eingelesen <===
19864  9133
19865  9133  20 45 a8   loadnp jsr mapio        I/O-Chips einblenden
19866  9136
19867  9136  a5 0c              lda bverck       Flag LOAD=0, VERIFY=1 (BASIC)
19868  9138  a6 2d              ldx txttab       ADDRL Untergrenze akt. PGM
19869  913a  a4 2e              ldy txttab+1     ADDRH Untergrenze akt. PGM
19870  913c  20 d5 ff           jsr kloads       load/verify (ac) ab Adresse in xr/yr
                                                 (Sprungv.)
19871  913f
19872  913f  08                 php
19873  9140  20 43 92           jsr dschk        für FA > 3 DS$ rücksetzen
19874  9143  28                 plp
19875  9144  b0 65              bcs jerxit       Behandlung I/O-Fehler
19876  9146
19877  9146  a5 0c              lda bverck       Flag LOAD=0, VERIFY=1 (BASIC)
19878  9148  f0 16              beq cld50
19879  914a
19880  914a  a2 1c              ldx #_ervfy      vorbesetzen »?verify error«
19881  914c
19882  914c  20 51 92           jsr creads       RS-/System-Status nach ac
19883  914f  29 10              and #%00010000   Bit4: 1 = verify error
19884  9151  d0 16              bne jerror       Behandlung Fehlermeldung
19885  9153
19886  9153  24 7f              bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
19887  9155  30 08              bmi cld20
19888  9157
19889  9157  20 81 92           jsr cprimm       folg. Text ausg./überspringen
19890  915a
19891  915a  0d 4f 4b           .byte _cr, 'ok', _cr, _eom
19891  915d  0d 00
19892  915f
19893  915f  60         cld20   rts
19894  9160
19895  9160
19896  9160  20 51 92   cld50   jsr creads       RS-/System-Status nach ac
19897  9163  29 bf              and #%10111111
19898  9165
19899  9165   7.......  1 = Dev.not pres. oder Band-Ende
19900  9165   .6......  1 = End of File: ausgeblendet
19901  9165   ..5.....  1 = Prüfsummenfehler
19902  9165   ...4....  1 = verify error
19903  9165   ....3...  1 = Block zu lang (Kass.)
19904  9165   .....2..  1 = Block zu kurz (Kass.)
19905  9165   ......1.  1 = Timeout Lesen
19906  9165   .......0  1 = Timeout Schreiben
19907  9165
19908  9165  f0 05              beq cld60        Lade-Endadr. speichern, BASIC
                                                 Eingabeschleife
19909  9167
19910  9167
19911  9167  ===> Behandlung »?load error« <===
19912  9167
19913  9167  a2 1d      loderr ldx #_erlod
19914  9169
19915  9169
```

```
19915  9169  ===> ind. Behandlung Fehlermeldung <===
19916  9169
19917  9169  4c 3c 4d   jerror jmp error        ind. Behandlung Fehlermeldung
19918  916c
19919  916c
19920  916c  ===> Lade-Endadr. speichern, BASIC Eingabeschleife <===
19921  916c
19922  916c  8e 10 12   cld60  stx txttop        ADDRL Obergrenze akt. PGM
19923  916f  8c 11 12          sty txttop+1      ADDRH Obergrenze akt. PGM
19924  9172
19925  9172  24 7f             bit runmod        Modus: $80=PGM, $40=OVL, $00=DIR
19926  9174  30 0e             bmi cld70         Overlay-LOAD
19927  9176  70 e7             bvs cld20
19928  9178
19929  9178  20 2a 4d          jsr reddy         Ausgabe »ready.«
19930  917b  20 4f 4f          jsr lnkprg        alle Linkadressen neu berechnen
19931  917e  20 f3 51          jsr runc          BASIC-Zeiger initialisieren, CLR
19932  9181  4c c3 4d          jmp main          ind. BASIC Eingabeschleife
19933  9184
19934  9184
19935  9184  ===> Overlay-LOAD <===
19936  9184
19937  9184  20 54 52   cld70  jsr stxtpt       txtptr auf PGM Anf.
19938  9187  20 4f 4f          jsr lnkprg        alle Linkadressen neu berechnen
19939  918a  4c 35 52          jmp fload         reset read-, Stapel- und
                                                 Befehlszeiger
19940  918d
19941  918d
19942  918d  ===> BASIC-Statement OPEN ($9f) <===
19943  918d
19944  918d  20 f6 91   open   jsr paoc          Parameter OPEN/CLOSE einlesen
19945  9190  18                clc
19946  9191  20 d8 90          jsr copen         BASIC-Statement OPEN ($9f)
19947  9194  20 43 92          jsr dschk         für FA > 3 DS$ rücksetzen
19948  9197  b0 12             bcs jerxit        Behandlung I/O-Fehler
19949  9199
19950  9199  60                rts
19951  919a
19952  919a
19953  919a  ===> BASIC-Statement CLOSE ($a0) <===
19954  919a
19955  919a  20 f6 91   cclos  jsr paoc          Parameter OPEN/CLOSE einlesen
19956  919d  20 45 a8          jsr mapio         I/O-Chips einblenden
19957  91a0
19958  91a0  a5 4b             lda lstpnt        la
19959  91a2  18                clc
19960  91a3  20 75 92          jsr cclose        log. File (la in ac) schließen
19961  91a6
19962  91a6  20 43 92          jsr dschk         für FA > 3 DS$ rücksetzen
19963  91a9  90 b4             bcc cld20
19964  91ab
19965  91ab  4c d0 90   jerxit jmp erexit        Behandlung I/O-Fehler
19966  91ae
19967  91ae
19968  91ae  a9 00      plsv   lda #0            FNLEN, LA
19969  91b0  20 5d 92          jsr csetnm        ac, xr/yr nach fnlen, fnadr/+1
19970  91b3
19971  91b3  a2 01             ldx #_cass        FA
```

```
19972  91b5  a0 00           ldy #0          SA
19973  91b7  20 57 92        jsr cstlfs      ac, xr, yr nach la, fa, sa
19974  91ba
19975  91ba  20 87 92        jsr cstbnk      PGM Bank
19976  91bd
19977  91bd  20 e3 91        jsr paoc20      Test weit. Param., sonst Abbruch
19978  91c0  20 39 92        jsr paoc15      Filenamen einlesen/eintragen
19979  91c3
19980  91c3  20 e3 91        jsr paoc20      Test weit. Param., sonst Abbruch
19981  91c6  20 dd 91        jsr plsv7       FA
19982  91c9
19983  91c9  a0 00           ldy #0          SA
19984  91cb  86 4b           stx lstpnt      FA
19985  91cd  20 57 92        jsr cstlfs      ac, xr, yr nach la, fa, sa
19986  91d0
19987  91d0  20 e3 91        jsr paoc20      Test weit. Param., sonst Abbruch
19988  91d3  20 dd 91        jsr plsv7       Test Komma, weit. Param. nach xr
19989  91d6  8a              txa
19990  91d7  a8              tay             SA
19991  91d8  a6 4b           ldx lstpnt      FA
19992  91da  4c 57 92        jmp cstlfs      ac, xr, yr nach la, fa, sa
19993  91dd
19994  91dd
19995  91dd  ===> Test Komma, weit. Param. nach xr <===
19996  91dd
19997  91dd  20 eb 91   plsv7 jsr paoc30      Test Komma, weit. Param., sonst
                                             »?syntax error«
19998  91e0  4c f4 87         jmp getbyt      8 Bit INT üb. frmnum nach xr
19999  91e3
20000  91e3
20001  91e3  ===> Test weit. Param., sonst Abbruch <===
20002  91e3
20003  91e3  20 86 03   paoc20 jsr chrgot      letztes Zeichen erneut nach ac
20004  91e6  d0 02            bne paocx
20005  91e8
20006  91e8  68               pla
20007  91e9  68               pla
20008  91ea  60         paocx rts
20009  91eb
20010  91eb
20011  91eb  ===> Test Komma, weit. Param., sonst »?syntax error« <===
20012  91eb
20013  91eb  20 5c 79   paoc30 jsr chkcom      Test auf folg. »Komma« sonst Fehler
20014  91ee
20015  91ee
20016  91ee  ===> »?syntax error«, wenn Trennzeichen <===
20017  91ee
20018  91ee  20 86 03   paoc32 jsr chrgot      letztes Zeichen erneut nach ac
20019  91f1  d0 f7            bne paocx
20020  91f3
20021  91f3  4c 6c 79         jmp snerr       Behandlung »?syntax error«
20022  91f6
20023  91f6
20024  91f6  ===> Parameter OPEN/CLOSE einlesen <===
20025  91f6
20026  91f6  a9 00      paoc  lda #_pgmbk     akt. Bank Speicher-Operation
20027  91f8  a2 01            ldx #_varbk     Bank des akt. Filenamens
```

```
20028  91fa  20 87 92          jsr cstbnk       ac nach RAM-Bank, xr nach
                                                Filename-Bank
20029  91fd
20030  91fd  20 5d 92          jsr csetnm       FNLEN = 0
20031  9200
20032  9200  20 ee 91          jsr paoc32       »?syntax error«, wenn Trennzeichen
20033  9203  20 f4 87          jsr getbyt       8 Bit INT üb. frmnum nach xr
20034  9206  86 4b             stx lstpnt       LA
20035  9208
20036  9208  8a                txa              LA
20037  9209  a2 01             ldx #_cass       FA
20038  920b  a0 00             ldy #0           SA
20039  920d  20 57 92          jsr cstlfs       vorbesetzen
20040  9210
20041  9210  20 e3 91          jsr paoc20       Test weit. Param., sonst Abbruch
20042  9213  20 dd 91          jsr plsv7        Test Komma, weit. Param. nach xr
20043  9216  86 4c             stx lstpnt+1     FA
20044  9218
20045  9218  a0 00             ldy #0           SA
20046  921a  a5 4b             lda lstpnt       LA
20047  921c  e0 03             cpx #_scrn       FA Bildschirm
20048  921e  90 01             bcc paoc5
20049  9220
20050  9220  88                dey              SA = 255
20051  9221  20 57 92   paoc5  jsr cstlfs       ac, xr, yr nach la, fa, sa
20052  9224
20053  9224  20 e3 91          jsr paoc20       Test weit. Param., sonst Abbruch
20054  9227  20 dd 91          jsr plsv7        Test Komma, weit. Param. nach xr
20055  922a  8a                txa
20056  922b  a8                tay              SA
20057  922c  a6 4c             ldx lstpnt+1     FA
20058  922e  a5 4b             lda lstpnt       LA
20059  9230  20 57 92          jsr cstlfs       ac, xr, yr nach la, fa, sa
20060  9233
20061  9233  20 e3 91          jsr paoc20       Test weit. Param., sonst Abbruch
20062  9236  20 eb 91          jsr paoc30       Test Komma, weit. Param., sonst
                                                »?syntax error«
20063  9239
20064  9239
20065  9239  ===> Filenamen einlesen/eintragen <===
20066  9239
20067  9239  20 7b 87   paoc15 jsr sav13        String Filename einlesen
20068  923c  a6 24             ldx index1       ADDRL Filename
20069  923e  a4 25             ldy index1+1     ADDRH Filename
20070  9240  4c 5d 92          jmp csetnm       ac, xr/yr nach fnlen, fnadr/+1
20071  9243
20072  9243
20073  9243  ===> für FA > 3 DS$ rücksetzen <===
20074  9243
20075  9243  08         dschk  php
20076  9244  48                pha
20077  9245
20078  9245  a5 ba             lda fa           akt. Primäradresse
20079  9247  c9 04             cmp #_scrn+1
20080  9249  90 03             bcc dsch10
20081  924b
20082  924b  20 0d a8          jsr oldclr       DS$-String ungültig machen
20083  924e  68         dsch10 pla
```

```
20084  924f  28              plp
20085  9250  60              rts
20086  9251
20087  9251
20088  9251  ===> RS-/System-Status nach ac <===
20089  9251
20090  9251  20 45 a8   creads jsr mapio       I/O-Chips einblenden
20091  9254  4c b7 ff         jmp kreads       RS-/System-Status nach ac (Sprungv.)
20092  9257
20093  9257
20094  9257  ===> ac, xr, yr nach la, fa, sa <===
20095  9257
20096  9257  20 45 a8   cstlfs jsr mapio       I/O-Chips einblenden
20097  925a  4c ba ff         jmp kstlfs       ac, xr, yr nach la, fa, sa
                                               (Sprungv.)
20098  925d
20099  925d
20100  925d  ===> ac, xr/yr nach fnlen, fnadr/+1 <===
20101  925d
20102  925d  20 45 a8   csetnm jsr mapio       I/O-Chips einblenden
20103  9260  4c bd ff         jmp ksetnm       ac, xr/yr nach fnlen, fnadr/+1
                                               (Sprungv.)
20104  9263
20105  9263
20106  9263  ===> ind. Zeich. akt. Kanal nach ac <===
20107  9263
20108  9263  20 45 a8   cbasin jsr mapio       I/O-Chips einblenden
20109  9266  4c cf ff          jmp kbasin       ind. Zeich. akt. Kanal nach ac
                                               (Sprungv.)
20110  9269
20111  9269
20112  9269  ===> ind. Ausgabe ac auf akt. Kanal <===
20113  9269
20114  9269  20 45 a8   cbsout jsr mapio       I/O-Chips einblenden
20115  926c  4c d2 ff         jmp kbsout       ind. Ausg. ac auf akt. Kanal
                                               (Sprungv.)
20116  926f
20117  926f
20118  926f  ===> ind. I/O-Kanäle schließen <===
20119  926f
20120  926f  20 45 a8   cclrch jsr mapio       I/O-Chips einblenden
20121  9272  4c cc ff         jmp kclrch       ind. I/O-Kanäle schließen (Sprungv.)
20122  9275
20123  9275
20124  9275  ===> log. File (la in ac) schließen <===
20125  9275
20126  9275  20 45 a8   cclose jsr mapio       I/O-Chips einblenden
20127  9278  4c c3 ff         jmp kclose       log. File (la in ac) schließen
                                               (Sprungv.)
20128  927b
20129  927b
20130  927b  ===> off. Files :=0, Ersatz-I/O <===
20131  927b
20132  927b  20 45 a8   cclall jsr mapio       I/O-Chips einblenden
20133  927e  4c e7 ff         jmp kclall       off. Files :=0, Ersatz-I/O
                                               (Sprungv.)
20134  9281
20135  9281
```

```
20136  9281  ===> folg. Text ausg./überspringen <===
20137  9281
20138  9281  20 45 a8   cprimm jsr mapio        I/O-Chips einblenden
20139  9284  4c 7d ff          jmp kprimm       folg. Text ausg./überspringen
                                                (Sprungv.)
20140  9287
20141  9287
20142  9287  ===> ac nach RAM-Bank, xr nach Filename-Bank <===
20143  9287
20144  9287  20 45 a8   cstbnk jsr mapio        I/O-Chips einblenden
20145  928a  4c 68 ff          jmp kstbnk       ac nach RAM-Bank, xr nach
                                                Filename-Bank (Sprungv.)
20146  928d
20147  928d
20148  928d  ===> Cursor-Pos. (rel. BS-Fenster) les./schreib. <===
20149  928d
20150  928d  8d 03 ff   cplot  sta lcrc         ROM / PGM-RAM einschalten
20151  9290
20152  9290  4c f0 ff          jmp kplot        Cursor-Pos. (rel. BS-Fenster)
                                                les./schreib. (Sprungv.)
20153  9293
20154  9293
20155  9293  ===> Abfrage Stopp-Taste, dann Ersatz-I/O <===
20156  9293
20157  9293  20 45 a8   cstop  jsr mapio        I/O-Chips einblenden
20158  9296  4c e1 ff          jmp kstop        Abfrage Stopp-Taste, dann Ersatz-I/O
                                                (Sprungv.)
20159  9299
20160  9299
20161  9299                    .end
20162  9299                    .lib basic.garbcol
20163  9299  put"@basic.garbcol"
```

```
20165  9299
20166  9299
20167  9299  ===> Arbeitsstring Länge ac anlegen <===
20168  9299
20169  9299  46 11      getspa lsr dores        Flag Garb.Coll.
20170  929b
20171  929b   7.......  0 = Garb.Coll. noch nicht durchgeführt
20172  929b
20173  929b  aa         tryag2 tax              Länge Arb.String
20174  929c  f0 3b             beq getrts
20175  929e
20176  929e  48                 pha             Länge Arb.String
20177  929f
20178  929f  ===> Zeiger auf Trailer Arb.String <===
20179  929f
20180  929f  a5 35             lda fretop      ADDRL Zeiger Anfang String-Variable
20181  92a1  38                 sec
20182  92a2  e9 02             sbc #2          Länge Trailer
20183  92a4
20184  92a4  a4 36             ldy fretop+1    ADDRH Zeiger Anfang String-Variable
20185  92a6  b0 01             bcs tryag3
20186  92a8  88                 dey
20187  92a9
20188  92a9  85 24      tryag3 sta index1      ADDRL Zeiger auf Trailer
20189  92ab  84 25             sty index1+1    ADDRH Zeiger auf Trailer
20190  92ad
20191  92ad  ===> Zeiger Anf. Arb.String berechnen, Test/Garb.Coll. <===
20192  92ad
20193  92ad  8a                 txa             Länge Arb.String
20194  92ae  49 ff             eor #$ff
20195  92b0  38                 sec
20196  92b1  65 24             adc index1
20197  92b3  b0 01             bcs tryag4
20198  92b5  88                 dey
20199  92b6
20200  92b6  c4 34      tryag4 cpy strend+1    ADDRH+1 Obergr. ARRAY Tabelle
20201  92b8  90 20             bcc garbag      Test / Garbage Collection
20202  92ba  d0 04             bne strfre      Trailer Arb.String, fretop eintragen
20203  92bc
20204  92bc  c5 33             cmp strend      ADDRL+1 Obergr. ARRAY Tabelle
20205  92be  90 1a             bcc garbag      Test / Garbage Collection
20206  92c0
20207  92c0
20208  92c0  ===> Trailer Arb.String, fretop eintragen <===
20209  92c0
20210  92c0  85 37      strfre sta frespc      ADDRL Zeiger auf neuen String
20211  92c2  84 38             sty frespc+1    ADDRH Zeiger auf neuen String
20212  92c4
20213  92c4  a0 01             ldy #1          Zeiger auf 2. Trailer Byte
20214  92c6  a9 ff             lda #$ff        Flag für ungült. String
20215  92c8  8d 04 ff          sta lcrd        ROM / VAR-RAM einschalten
20216  92cb  91 24             sta (index1),y  Zeiger auf Trailer Arb.String
20217  92cd
20218  92cd  88                 dey             Zeiger auf 1. Trailer Byte
20219  92ce  68                 pla             Länge Arb.String
20220  92cf  91 24             sta (index1),y  Zeiger auf Trailer Arb.String
20221  92d1
20222  92d1  a6 37             ldx frespc      ADDRL Zeiger auf neuen String
```

```
20223  92d3  a4 38           ldy frespc+1     ADDRH Zeiger auf neuen String
20224  92d5
20225  92d5  86 35           stx fretop       ADDRL Zeiger Anfang String-Variable
20226  92d7  84 36           sty fretop+1     ADDRH Zeiger Anfang String-Variable
20227  92d9
20228  92d9  60        getrts rts
20229  92da
20230  92da
20231  92da  ===> Test / Garbage Collection <===
20232  92da
20233  92da  a5 11      garbag lda dores       Flag für DATA, LIST, garbage
                                               collection
20234  92dc
20235  92dc   7.......  1 = Garb.Coll. bereits durchgeführt
20236  92dc
20237  92dc  30 09           bmi grbg99       Behandlung »?out of memory error«
20238  92de
20239  92de  ===> Garbage Collection durchführen <===
20240  92de
20241  92de  20 ea 92        jsr garba2       Garbage Collection
20242  92e1
20243  92e1  38              sec
20244  92e2  66 11           ror dores        Flag für DATA, LIST, garbage
                                               collection
20245  92e4
20246  92e4   7.......  1 = Garb.Coll. durchgeführt
20247  92e4
20248  92e4  68              pla              Länge Arb.String
20249  92e5  d0 b4           bne tryag2       immer
20250  92e7
20251  92e7
20252  92e7  ===> Behandlung »?out of memory error« <===
20253  92e7
20254  92e7  4c 3a 4d   grbg99 jmp omerr       Behandlung »?out of memory error«
20255  92ea
20256  92ea
20257  92ea  ===> Garbage Collection <===
20258  92ea
20259  92ea  a6 18      garba2 ldx temppt      Zeiger in den String-Stapel
20260  92ec
20261  92ec
20262  92ec  ===> alle Arb.Strings gültig machen <===
20263  92ec
20264  92ec  e0 1b      garb1  cpx #_tmpst
20265  92ee  f0 13           beq garb2        String Stapel leer
20266  92f0
20267  92f0  20 f0 93        jsr slr1         Zeig. Trailer String aus Stapel
                                               berechnen
20268  92f3  f0 f7           beq garb1        String Länge =0
20269  92f5
20270  92f5  ===> Adresse Stringdescr. nach Trailer <===
20271  92f5
20272  92f5  8a              txa              ADDRL String Descr.
20273  92f6  a0 00           ldy #0           Zeiger 1. Trailer Byte
20274  92f8  8d 04 ff        sta lcrd         ROM / VAR-RAM einschalten
20275  92fb  91 5e           sta (str1+1),y
20276  92fd
20277  92fd  98              tya              ADDRH String Descr.
```

```
20278  92fe  c8              iny             Zeiger 2. Trailer Byte
20279  92ff  91 5e           sta (str1+1),y
20280  9301  d0 e9           bne garb1       immer
20281  9303
20282  9303
20283  9303  a0 00     garb2  ldy #0
20284  9305  84 5a           sty highds
20285  9307
20286  9307  a6 39           ldx mxmem1      ADDRL Obergrenze VAR RAM
20287  9309  a4 3a           ldy mxmem1+1    ADDRH Obergrenze VAR RAM
20288  930b
20289  930b  86 61           stx grbtop      ADDRL Ziel Zeiger
20290  930d  86 50           stx grbpnt      ADDRL Quelle Zeiger
20291  930f  86 37           stx frespc      ADDRL Zeiger auf neuen String
20292  9311
20293  9311  84 62           sty grbtop+1    ADDRH Ziel Zeiger
20294  9313  84 51           sty grbpnt+1    ADDRH Quelle Zeiger
20295  9315  84 38           sty frespc+1    ADDRH Zeiger auf neuen String
20296  9317
20297  9317  8a              txa             mxmem
20298  9318
20299  9318
20300  9318  ===> nächsten gült. String suchen <===
20301  9318
20302  9318  20 83 93  gloop  jsr chkgrb      Test nächst. String gültig, Addr.
                                             Descr.
20303  931b  d0 0c           bne col01
20304  931d
20305  931d
20306  931d  ===> ung. String, Quelle Zeig. nachf./Flag <===
20307  931d
20308  931d  88        col00a dey             Zeiger 1. Trailer Byte (Länge)
20309  931e  20 fb 42         jsr indgrb      LDA (grbpnt),y aus VAR-RAM
20310  9321  20 d2 93         jsr movpnt      ac von Quelle Zeiger subtrahieren
20311  9324
20312  9324  38              sec
20313  9325  66 5a           ror highds
20314  9327
20315  9327   7.......  1 = ungült. String gefunden
20316  9327
20317  9327  d0 ef           bne gloop       immer
20318  9329
20319  9329
20320  9329  24 5a     col01  bit highds
20321  932b  10 42           bpl col03       gültige Strings überspringen
20322  932d
20323  932d  a2 00           ldx #0
20324  932f  86 5a           stx highds
20325  9331
20326  9331  a9 02           lda #2          ********* ? BUG ? **********
20327  9333
20328  9333
20329  9333  ===> gültigen String über ungültigen kopieren <===
20330  9333
20331  9333  a0 01     col02  ldy #1          Zeiger 2. Trailer Byte
20332  9335  20 fb 42         jsr indgrb      LDA (grbpnt),y aus VAR-RAM
20333  9338  91 61           sta (grbtop),y  Ziel
20334  933a
```

```
20335  933a  88              dey             Zeiger 1. Trailer Byte
20336  933b  20 fb 42        jsr indgrb      LDA (grbpnt),y aus VAR-RAM
20337  933e  91 61           sta (grbtop),y  Ziel
20338  9340
20339  9340  20 b7 03        jsr ind1_1      LDA (index1),y aus VAR-RAM
20340  9343
20341  9343  aa              tax             String Länge
20342  9344  20 e1 93        jsr movtop      ac von Ziel Zeiger subtrahieren
20343  9347
20344  9347  85 37           sta frespc      ADDRL Zeiger auf neuen String
20345  9349  84 38           sty frespc+1    ADDRH Zeiger auf neuen String
20346  934b
20347  934b  8a              txa             String Länge
20348  934c  20 d2 93        jsr movpnt      ac von Quelle Zeiger subtrahieren
20349  934f
20350  934f  8a              txa             String Länge
20351  9350  a8              tay             String Länge
20352  9351
20353  9351  88        glop1 dey             Zeiger in String
20354  9352  20 fb 42        jsr indgrb      LDA (grbpnt),y aus VAR-RAM
20355  9355  91 61           sta (grbtop),y  Ziel
20356  9357
20357  9357  ca              dex             String Länge
20358  9358  d0 f7           bne glop1
20359  935a
20360  935a  ===> String Adresse nach Descr. <===
20361  935a
20362  935a  a0 02           ldy #2
20363  935c  b9 60 00  col02b lda grbtop-1,y  String Adresse
20364  935f  91 24           sta (index1),y
20365  9361  88              dey
20366  9362  d0 f8           bne col02b
20367  9364
20368  9364  a5 50           lda grbpnt      ADDRL Quelle Zeiger
20369  9366  a4 51           ldy grbpnt+1    ADDRH Quelle Zeiger
20370  9368  20 83 93        jsr chkgrb      Test nächst. String gültig, Addr.
                                             Descr.
20371  936b
20372  936b  f0 b0           beq col00a      ung. String, Quelle Zeig. nachf./Flag
20373  936d  d0 c4           bne col02       gültigen String über ungültigen
                                             kopieren
20374  936f
20375  936f
20376  936f  ===> gültige Strings überspringen <===
20377  936f
20378  936f  a0 00      col03 ldy #0          Zeiger String Länge Descr.
20379  9371  20 b7 03        jsr ind1_1      LDA (index1),y aus VAR-RAM
20380  9374
20381  9374  aa              tax             String Länge
20382  9375  20 e1 93        jsr movtop      ac von Ziel Zeiger subtrahieren
20383  9378
20384  9378  85 37           sta frespc      ADDRL Zeiger auf neuen String
20385  937a  84 38           sty frespc+1    ADDRH Zeiger auf neuen String
20386  937c
20387  937c  8a              txa             String Länge
20388  937d  20 d2 93        jsr movpnt      ac von Quelle Zeiger subtrahieren
20389  9380  4c 18 93        jmp gloop       nächsten gült. String suchen
20390  9383
```

```
20391  9383
20392  9383
20392  9383  ===> Test nächst. String gültig, Addr. Descr. <===
20393  9383
20394  9383  c4 36      chkgrb cpy fretop+1     ADDRH Zeiger Anfang String-Variable
20395  9385  90 2a             bcc cfre4        alle Arb.Strings ungültig machen
20396  9387  d0 06             bne cfre1
20397  9389
20398  9389  c5 35             cmp fretop       ADDRL Zeiger Anfang String-Variable
20399  938b  f0 24             beq cfre4        alle Arb.Strings ungültig machen
20400  938d  90 22             bcc cfre4        alle Arb.Strings ungültig machen
20401  938f
20402  938f  24 5a      cfre1  bit highds
20403  9391
20404  9391   7.......  1 = ungült. String gefunden
20405  9391
20406  9391  30 05             bmi cfre2
20407  9393
20408  9393  a9 02             lda #2           Trailer Bytes
20409  9395  20 e1 93          jsr movtop       ac von Ziel Zeiger subtrahieren
20410  9398
20411  9398  a9 02      cfre2  lda #2           Trailer Bytes
20412  939a  20 d2 93          jsr movpnt       ac von Quelle Zeiger subtrahieren
20413  939d
20414  939d  a0 01             ldy #1           Zeiger auf 2. Trailer Byte
20415  939f  20 fb 42          jsr indgrb       LDA (grbpnt),y aus VAR-RAM
20416  93a2
20417  93a2  c9 ff             cmp #$ff         Flag für ungült. String
20418  93a4  d0 01             bne cfre3        Adr. Descr. aus Trailer nach index1
20419  93a6
20420  93a6  60                rts
20421  93a7
20422  93a7
20423  93a7  ===> Adr. Descr. aus Trailer nach index1 <===
20424  93a7
20425  93a7  20 fb 42   cfre3  jsr indgrb       LDA (grbpnt),y aus VAR-RAM
20426  93aa  99 24 00          sta index1,y
20427  93ad  88                dey
20428  93ae  10 f7             bpl cfre3        Adr. Descr. aus Trailer nach index1
20429  93b0
20430  93b0  60                rts
20431  93b1
20432  93b1
20433  93b1  ===> alle Arb.Strings ungültig machen <===
20434  93b1
20435  93b1  a6 18      cfre4  ldx temppt       Zeiger in den String-Stapel
20436  93b3
20437  93b3  e0 1b      cfre4a cpx #_tmpst
20438  93b5  f0 10             beq cfre4b       String Stapel leer
20439  93b7
20440  93b7  20 f0 93          jsr slr1         Zeig. Trailer String aus Stapel
                                                berechnen
20441  93ba  f0 f7             beq cfre4a       String Länge =0
20442  93bc
20443  93bc  a0 00             ldy #0           Zeiger 1. Trailer Byte
20444  93be  91 5e             sta (str1+1),y   String Länge
20445  93c0
20446  93c0  c8                iny              Zeiger 2. Trailer Byte
```

```
20447  93c1
20448  93c1  a9 ff           lda #$ff          Flag ungült. String
20449  93c3  91 5e           sta (str1+1),
20450  93c5  d0 ec           bne cfre4a        immer
20451  93c7
20452  93c7
20453  93c7  ===> neue String Untergrenze eintragen <===
20454  93c7
20455  93c7  68        cfre4b pla              RTS Adresse
20456  93c8  68               pla
20457  93c9
20458  93c9  a5 37            lda frespc        ADDRL Zeiger auf neuen String
20459  93cb  a4 38            ldy frespc+1      ADDRH Zeiger auf neuen String
20460  93cd
20461  93cd  85 35            sta fretop        ADDRL Zeiger Anfang String-Variable
20462  93cf  84 36            sty fretop+1      ADDRH Zeiger Anfang String-Variable
20463  93d1
20464  93d1  60               rts
20465  93d2
20466  93d2
20467  93d2  ===> ac von Quelle Zeiger subtrahieren <===
20468  93d2
20469  93d2  49 ff      movpnt eor #$ff
20470  93d4  38               sec
20471  93d5  65 50            adc grbpnt        ADDRL Quelle Zeiger
20472  93d7
20473  93d7  a4 51            ldy grbpnt+1      ADDRH Quelle Zeiger
20474  93d9  b0 01            bcs mov00
20475  93db  88               dey
20476  93dc
20477  93dc  85 50      mov00 sta grbpnt        ADDRL Quelle Zeiger
20478  93de  84 51            sty grbpnt+1      ADDRH Quelle Zeiger
20479  93e0
20480  93e0  60               rts
20481  93e1
20482  93e1
20483  93e1  ===> ac von Ziel Zeiger subtrahieren <===
20484  93e1
20485  93e1  49 ff      movtop eor #$ff
20486  93e3  38               sec
20487  93e4  65 61            adc grbtop        ADDRL Ziel Zeiger
20488  93e6
20489  93e6  a4 62            ldy grbtop+1      ADDRH Ziel Zeiger
20490  93e8  b0 01            bcs mov01
20491  93ea  88               dey
20492  93eb
20493  93eb  85 61      mov01 sta grbtop        ADDRL Ziel Zeiger
20494  93ed  84 62            sty grbtop+1      ADDRH Ziel Zeiger
20495  93ef
20496  93ef  60               rts
20497  93f0
20498  93f0
20499  93f0  ===> Zeig. Trailer String aus Stapel berechnen <===
20500  93f0
20501  93f0  ca          slr1 dex               Zeiger in String Stapel
20502  93f1  b5 00            lda 0,x           ADDRH String
20503  93f3  85 5f            sta str1+2
20504  93f5
```

```
20505  93f5  ca               dex             Zeiger in String Stapel
20506  93f6  b5 00            lda 0,x         ADDRL String
20507  93f8  85 5e            sta str1+1
20508  93fa
20509  93fa  ca               dex             Zeiger in String Stapel
20510  93fb  b5 00            lda 0,x         Länge String
20511  93fd  48               pha             Länge String
20512  93fe
20513  93fe  18               clc
20514  93ff  65 5e            adc str1+1      ADDRL String
20515  9401  85 5e            sta str1+1      ADDRL Trailer
20516  9403
20517  9403  90 02            bcc slr2
20518  9405  e6 5f            inc str1+2      ADDRH Trailer
20519  9407
20520  9407  68         slr2  pla             Länge String
20521  9408  60               rts
20522  9409                    20523  9409
20524  9409                    .end
20525  9409                    .lib basic.math_5
20526  9409  put@basic.math_5
```

```
20528  9409
20529  9409
20530  9409  ===> BASIC-Funktion COS ($be) <===
20531  9409
20532  9409  a9 85      cos    lda #<pi2        Konstante pi/2 = 1.57079633
20533  940b  a0 94             ldy #>pi2        Konstante pi/2 = 1.57079633
20534  940d  20 12 8a          jsr romadd       ROM Konst. ac/yr zu FAC addieren
20535  9410
20536  9410
20537  9410  ===> BASIC-Funktion SIN ($bf) <===
20538  9410
20539  9410  20 38 8c  sin    jsr movaf        FAC runden, FAC nach ARG
20540  9413
20541  9413  a9 8a             lda #<twopi      Konstante 2*pi = 6.28318531
20542  9415  a0 94             ldy #>twopi      Konstante 2*pi = 6.28318531
20543  9417  a6 6f             ldx argsgn       ARG: SG
20544  9419  20 41 8b          jsr fdivf        FAC durch VAR ac/yr divid.
20545  941c  20 38 8c          jsr movaf        FAC runden, FAC nach ARG
20546  941f  20 fb 8c          jsr int          BASIC-Funktion INT ($b5)
20547  9422  a9 00             lda #0
20548  9424  85 70             sta arisgn       Vorzeichen Ergebnis Operation
20549  9426
20550  9426  20 31 88          jsr fsubt        BASIC-Funktion MINUS ($ab)
20551  9429  a9 8f             lda #<fr4        Konstante 0.25
20552  942b  a0 94             ldy #>fr4        Konstante 0.25
20553  942d  20 18 8a          jsr romsub       ROM Konst. ac/yr von FAC subtr.
20554  9430
20555  9430  a5 68             lda facsgn       FAC: SG
20556  9432  48               pha
20557  9433
20558  9433  10 0d            bpl sin1
20559  9435
20560  9435  20 0e 8a          jsr faddh        0.5 zu FAC addieren
20561  9438  a5 68             lda facsgn       FAC: SG
20562  943a  30 09            bmi sin2
20563  943c
20564  943c  a5 14             lda tansgn       Flag für TAN-Vorzeichen
20565  943e  49 ff             eor #$ff
20566  9440  85 14             sta tansgn       Flag für TAN-Vorzeichen
20567  9442
20568  9442  20 fa 8f  sin1    jsr negop        BASIC-Funktion NEG. VORZEICHEN
20569  9445
20570  9445  a9 8f      sin2   lda #<fr4        Konstante 0.25
20571  9447  a0 94             ldy #>fr4        Konstante 0.25
20572  9449  20 12 8a          jsr romadd       ROM Konst. ac/yr zu FAC addieren
20573  944c
20574  944c  68               pla
20575  944d  10 03            bpl sin3
20576  944f
20577  944f  20 fa 8f          jsr negop        BASIC-Funktion NEG. VORZEICHEN
20578  9452
20579  9452  a9 94      sin3   lda #<sincon     Grad (5) für Polynom SIN/COSá
20580  9454  a0 94             ldy #>sincon     Grad (5) für Polynom SIN/COSá
20581  9456  4c 86 90          jmp polyx        Polynom-Auswertung
20582  9459
20583  9459
20584  9459
```

```
20584  9459  ===> BASIC-Funktion TAN ($c0) <===
20585  9459
20586  9459  20 fc 8b   tan     jsr mov1f         FAC runden, nach TMP FAC1
20587  945c
20588  945c  a9 00              lda #0            Flag PLUS
20589  945e  85 14              sta tansgn        Flag für TAN-Vorzeichen
20590  9460
20591  9460  20 10 94           jsr sin           BASIC-Funktion SIN ($bf)
20592  9463
20593  9463  a2 50              ldx #<tempf3
20594  9465  a0 00              ldy #>tempf3
20595  9467  20 00 8c           jsr movmf         FAC runden, nach RAM xr/yr
20596  946a
20597  946a  a9 59              lda #<tempf1      TMP FAC1
20598  946c  a0 00              ldy #>tempf1      TMP FAC1
20599  946e  20 d4 8b           jsr movfm         RAM Var. ac/yr nach FAC
20600  9471
20601  9471  a9 00              lda #0
20602  9473  85 68              sta facsgn        FAC: SG
20603  9475
20604  9475  a5 14              lda tansgn        Flag für TAN-Vorzeichen
20605  9477  20 81 94           jsr cosc
20606  947a
20607  947a  a9 50              lda #<tempf3
20608  947c  a0 00              ldy #>tempf3
20609  947e  4c 49 8b           jmp fdiv          RAM Var. ac/yr durch FAC dividieren
20610  9481
20611  9481
20612  9481  48         cosc    pha
20613  9482  4c 42 94           jmp sin1
20614  9485
20615  9485
20616  9485  ===> Konstante pi/2 = 1.57079633 <===
20617  9485
20618  9485  81 49 0f   pi2     .byte $81, $49, $0f, $da, $a2
20618  9488  da a2
20619  948a
20620  948a
20621  948a  ===> Konstante 2*pi = 6.28318531 <===
20622  948a
20623  948a  83 49 0f   twopi   .byte $83, $49, $0f, $da, $a2
20623  948d  da a2
20624  948f
20625  948f
20626  948f  ===> Konstante 0.25 <===
20627  948f
20628  948f  7f 00 00   fr4     .byte $7f, $00, $00, $00, $00 20628  9492  00 00
20629  9494
20630  9494
20631  9494  ===> Grad (5) für Polynom SIN/COSá <===
20632  9494
20633  9494  05         sincon .byte $05
20634  9495
20635  9495
20636  9495
20636  9495  ===> GK-Konstante: -14.3813907 <===
20637  9495
20638  9495  84 e6 1a   sinco5 .byte $84, $e6, $1a, $2d, $1b
```

```
20638  9498  2d 1b
20639  949a
20640  949a
20641  949a  ===> GK-Konstante:  42.0077971 <===
20642  949a
20643  949a  86 28 07   sinco4 .byte $86, $28, $07, $fb, $f8
20643  949d  fb f8
20644  949f
20645  949f
20646  949f  ===> GK-Konstante: -76.7041703 <===
20647  949f
20648  949f  87 99 68   sinco3 .byte $87, $99, $68, $89, $01
20648  94a2  89 01
20649  94a4
20650  94a4
20651  94a4  ===> GK-Konstante:  81.6052237 <===
20652  94a4
20653  94a4  87 23 35   sinco2 .byte $87, $23, $35, $df, $e1
20653  94a7  df e1
20654  94a9
20655  94a9
20656  94a9  ===> GK-Konstante: -41.3417021 <===
20657  94a9
20658  94a9  86 a5 5d   sinco1 .byte $86, $a5, $5d, $e7, $28
20658  94ac  e7 28
20659  94ae
20660  94ae
20661  94ae  ===> GK-Konstante:   6.28318531 <===
20662  94ae
20663  94ae  83 49 0f   sinco0 .byte $83, $49, $0f, $da, $a2
20663  94b1  da a2
20664  94b3
20665  94b3
20666  94b3  ===> BASIC-Funktion ATN ($c1) <===
20667  94b3
20668  94b3  a5 68      atn    lda facsgn       FAC: SG
20669  94b5  48               pha
20670  94b6  10 03            bpl atn1
20671  94b8
20672  94b8  20 fa 8f         jsr negop        BASIC-Funktion NEG. VORZEICHEN
20673  94bb
20674  94bb  a5 63      atn1   lda facexp       FAC: EX
20675  94bd  48               pha
20676  94be
20677  94be  c9 81            cmp #_expcn
20678  94c0  90 07            bcc atn2
20679  94c2
20680  94c2  a9 9c            lda #<fone       GK-Konstante 1.0
20681  94c4  a0 89            ldy #>fone       GK-Konstante 1.0
20682  94c6  20 1e 8a         jsr romdiv       ROM Konst. ac/yr durch FAC div.
20683  94c9
20684  94c9  a9 e3      atn2   lda #<atncon     Grad (11) für Polynom ATN
20685  94cb  a0 94            ldy #>atncon     Grad (11) für Polynom ATN
20686  94cd  20 86 90         jsr polyx        Polynom-Auswertung
20687  94d0
20688  94d0  68               pla
20689  94d1  c9 81            cmp #_expcn
20690  94d3  90 07            bcc atn3
```

```
20691  94d5
20692  94d5  a9 85           lda #<pi2       Konstante pi/2 = 1.57079633
20693  94d7  a0 94           ldy #>pi2       Konstante pi/2 = 1.57079633
20694  94d9  20 18 8a        jsr romsub      ROM Konst. ac/yr von FAC subtr.
20695  94dc
20696  94dc  68         atn3  pla
20697  94dd  10 03           bpl atn4
20698  94df
20699  94df  4c fa 8f        jmp negop       BASIC-Funktion NEG. VORZEICHEN
20700  94e2
20701  94e2
20702  94e2  60         atn4  rts
20703  94e3
20704  94e3
20705  94e3  ===> Grad (11) für Polynom ATN <===
20706  94e3
20707  94e3  0b         atncon .byte $0b
20708  94e4
20709  94e4
20710  94e4  ===> GK-Konstante: - 6.84793912 e-04 <===
20711  94e4
20712  94e4  76 b3 83   atnc11 .byte $76, $b3, $83, $bd, $d3
20712  94e7  bd d3
20713  94e9
20714  94e9
20715  94e9  ===> GK-Konstante:   4.85094216 e-03 <===
20716  94e9
20717  94e9  79 1e f4   atnc10 .byte $79, $1e, $f4, $a6, $f5
20717  94ec  a6 f5
20718  94ee
20719  94ee
20720  94ee  ===> GK-Konstante: - 0.0161117018 <===
20721  94ee
20722  94ee  7b 83 fc   atnc09 .byte $7b, $83, $fc, $b0, $10
20722  94f1  b0 10
20723  94f3
20724  94f3
20725  94f3  ===> GK-Konstante:   0.034209638 <===
20726  94f3
20727  94f3  7c 0c 1f   atnc08 .byte $7c, $0c, $1f, $67, $ca
20727  94f6  67 ca
20728  94f8
20729  94f8
20730  94f8  ===> GK-Konstante: - 0.0542791328 <===
20731  94f8
20732  94f8  7c de 53   atnc07 .byte $7c, $de, $53, $cb, $c1
20732  94fb  cb c1
20733  94fd
20734  94fd
20735  94fd  ===> GK-Konstante:   0.0724571965 <===
20736  94fd
20737  94fd  7d 14 64   atnc06 .byte $7d, $14, $64, $70, $4c 20737  9500  70 4c
20738  9502
20739  9502                    20740  9502
20740  9502  ===> GK-Konstante: - 0.0898023945 <===
20741  9502
20742  9502  7d b7 ea   atnc05 .byte $7d, $b7, $ea, $51, $7a
20742  9505  51 7a
```

```
20743  9507
20744  9507
20745  9507  ===> GK-Konstante:   0.110932413 <===
20746  9507
20747  9507  7d 63 30   atnc04 .byte $7d, $63, $30, $88, $7e 20747  950a  88 7e
20748  950c
20749  950c
20750  950c  ===> GK-Konstante: - 0.142839808 <===
20751  950c
20752  950c  7e 92 44   atnc03 .byte $7e, $92, $44, $99, $3a 20752  950f  99 3a
20753  9511
20754  9511
20755  9511  ===> GK-Konstante:   0.19999912 <===
20756  9511
20757  9511  7e 4c cc   atnc02 .byte $7e, $4c, $cc, $91, $c7
20757  9514  91 c7
20758  9516
20759  9516
20760  9516  ===> GK-Konstante: - 0.333333316 <===
20761  9516
20762  9516  7f aa aa   atnc01 .byte $7f, $aa, $aa, $aa, $13 20762  9519  aa 13
20763  951b
20764  951b
20765  951b  ===> GK-Konstante:   1 <===
20766  951b
20767  951b  81 00 00   atnc00 .byte $81, $00, $00, $00, $00 20767  951e  00 00
20768  9520
20769  9520
20770  9520                  .end
20771  9520                  .lib basic.using
20772  9520  put@basic.using
```

```
20774  9520
20775  9520
20776  9520  a2 ff      using  ldx #$ff
20777  9522  8e 36 01          stx endfd
20778  9525
20779  9525  ===> Zeiger auf Descr. Formatstring <===
20780  9525
20781  9525  20 80 03          jsr chrget       nächstes Zeichen nach ac
20782  9528  20 ef 77          jsr frmevl       folgenden Ausdruck auswerten
20783  952b  20 dd 77          jsr chkstr       Test String VAR, sonst type mism.
20784  952e
20785  952e  a5 66             lda facmo        ADDRL Descr.
20786  9530  48                pha
20787  9531
20788  9531  a5 67             lda faclo        ADDRH Descr.
20789  9533  48                pha
20790  9534
20791  9534  ===> Zeiger auf Formatstring <===
20792  9534
20793  9534  a0 02             ldy #2           Zeiger auf ADDRH Descr.
20794  9536  20 e7 42   ldform jsr indfmo       LDA (facmo),y aus VAR-RAM
20795  9539  88                dey
20796  953a  99 3f 00          sta fndpnt,y     ADDR Formatstring
20797  953d  d0 f7             bne ldform
20798  953f
20799  953f  20 e7 42          jsr indfmo       LDA (facmo),y aus VAR-RAM
20800  9542  8d 35 01          sta lfor         PRINT USING : Länge Formatstring
20801  9545
20802  9545  a8                tay
20803  9546  f0 0b             beq ser          Behandlung »?syntax error«
20804  9548
20805  9548
20806  9548  ===> Test auf # in Formatstring <===
20807  9548
20808  9548  88          cncj   dey
20809  9549  20 d3 42          jsr indfrm       LDA (fndpnt),y aus VAR-RAM
20810  954c  c9 23             cmp #'#'
20811  954e  f0 06             beq cscol
20812  9550
20813  9550  98                tya
20814  9551  d0 f5             bne cncj         Test auf # in Formatstring
20815  9553
20816  9553  ===> Behandlung »?syntax error« <===
20817  9553
20818  9553  4c 6c 79   ser    jmp snerr        Behandlung »?syntax error«
20819  9556
20820  9556
20821  9556  a9 3b      cscol  lda #';'
20822  9558  20 5e 79   eex2   jsr synchr       Test auf Zeichen in ac, sonst Fehler
20823  955b  84 77             sty mltus1
20824  955d  8c 23 01          sty bnr          PRINT USING : Zeiger auf Anfang
20825  9560
20826  9560  ===> Zahlenstring übernehmen <===
20827  9560
20828  9560  20 ef 77          jsr frmevl       folgenden Ausdruck auswerten
20829  9563  24 0f             bit valtyp       Flag Variablentyp
20830  9565  10 39             bpl conv         numer. VAR nach Arb.String
20831  9567
```

```
20832  9567  20 9f 97          jsr in
20833  956a  20 f2 98          jsr anaf
20834  956d  ae 2b 01          ldx chsn          PRINT USING : justify Flag
20835  9570  f0 15              beq prcha
20836  9572
20837  9572  a2 00              ldx #0
20838  9574  38                 sec
20839  9575  ad 31 01          lda cform         PRINT USING : Zeichen-Zähler
20840  9578  e5 78              sbc hulp          PRINT USING : Zähler
20841  957a  90 0b              bcc prcha
20842  957c
20843  957c  a2 3d              ldx #'='
20844  957e  ec 2b 01          cpx chsn          PRINT USING : justify Flag
20845  9581  d0 03              bne schs1
20846  9583
20847  9583  4a                 lsr a
20848  9584  69 00              adc #0
20849  9586  aa          schs1  tax
20850  9587  a0 00       prcha  ldy #0
20851  9589  8a          chx    txa
20852  958a  f0 05              beq cpef
20853  958c
20854  958c  ca                 dex
20855  958d  a9 20       oblk   lda #_spc
20856  958f  d0 08              bne outc          immer
20857  9591
20858  9591
20859  9591  c4 78       cpef   cpy hulp          PRINT USING : Zähler
20860  9593  b0 f8              bcs oblk
20861  9595
20862  9595  20 b7 03          jsr ind1_1        LDA (index1),y aus VAR-RAM
20863  9598  c8                 iny
20864  9599  20 eb 98    outc   jsr cdout
20865  959c  d0 eb              bne chx
20866  959e  f0 27              beq reay          immer
20867  95a0
20868  95a0
20869  95a0  ===> numer. VAR nach Arb.String <===
20870  95a0
20871  95a0  20 42 8e    conv   jsr fout          FAC nach String, Adr. nach ac/yr
20872  95a3
20873  95a3  a0 ff              ldy #$ff
20874  95a5  c8           conv1  iny
20875  95a6  b9 00 01          lda fbuffr,y
20876  95a9  d0 fa              bne conv1
20877  95ab
20878  95ab  98                 tya
20879  95ac  20 90 86          jsr strspa        Arb.string Länge ac anlegen
20880  95af
20881  95af  a0 00              ldy #0
20882  95b1  8d 04 ff          sta lcrd          ROM / VAR-RAM einschalten
20883  95b4
20884  95b4  b9 00 01    conv2  lda fbuffr,y
20885  95b7  f0 05              beq conv4
20886  95b9  91 64              sta (facho),y     FAC: M4
20887  95bb  c8                 iny
20888  95bc  d0 f6              bne conv2
20889  95be
```

```
20890  95be  20 e3 86  conv4  jsr putnew      String-Descriptor in Tabelle
20891  95c1  20 9f 97         jsr ini
20892  95c4  20 e7 95         jsr fform
20893  95c7  20 86 03  reay   jsr chrgot      letztes Zeichen erneut nach ac
20894  95ca  c9 2c             cmp #','
20895  95cc  f0 8a             beq eex2
20896  95ce
20897  95ce  38                sec
20898  95cf  66 77             ror mltus1
20899  95d1  20 f2 98          jsr anaf
20900  95d4  68                pla
20901  95d5  a8                tay
20902  95d6
20903  95d6  68                pla
20904  95d7  20 85 87          jsr fretmp      String ac/yr vom Stapel, ungültig
20905  95da  20 86 03          jsr chrgot      letztes Zeichen erneut nach ac
20906  95dd  c9 3b             cmp #';'
20907  95df  f0 03             beq nocr
20908  95e1
20909  95e1  4c 98 55          jmp crdo        Ausgabe cr (+lf) auf akt. Kanal
20910  95e4
20911  95e4
20912  95e4  4c 80 03  nocr   jmp chrget      nächstes Zeichen nach ac
20913  95e7
20914  95e7
20915  95e7  8d 03 ff  fform  sta lcrc        ROM / PGM-RAM einschalten
20916  95ea
20917  95ea  ad 04 12          lda pufill      PRINT USING: Füllzeichen
20918  95ed  8d 33 01          sta blfd        PRINT USING : Blank/Stern-Flag
20919  95f0
20920  95f0  a9 ff             lda #$ff
20921  95f2  8d 32 01  ana    sta sno         PRINT USING : Vorzeichennummer
20922  95f5
20923  95f5  4c fa 95          jmp insy
20924  95f8
20925  95f8
20926  95f8  86 80      stp    stx parsts      DOS Syntax Byte 1
20927  95fa
20928  95fa  c4 78      insy   cpy hulp        PRINT USING : Zähler
20929  95fc  f0 33             beq eoa
20930  95fe
20931  95fe  b9 00 01          lda fbuffr,y
20932  9601  c8                iny
20933  9602  c9 20             cmp #_spc
20934  9604  f0 f4             beq insy
20935  9606
20936  9606  c9 2d             cmp #'-'
20937  9608  f0 e8             beq ana
20938  960a
20939  960a  c9 2e             cmp #'.'
20940  960c  f0 ea             beq stp
20941  960e
20942  960e  c9 45             cmp #'e'
20943  9610  f0 11             beq lsg
20944  9612
20945  9612  9d 00 01          sta fbuffr,x
20946  9615  8e 24 01          stx enr         PRINT USING : Zeiger auf Ende
20947  9618  e8                inx
```

```
20948  9619
20949  9619  24 80          bit parsts      DOS Syntax Byte 1
20950  961b  10 dd          bpl insy
20951  961d
20952  961d  ee 2a 01       inc vn          PRINT USING : Anzahl Stellen vor
                                             Dezimalpunkt
20953  9620  4c fa 95       jmp insy
20954  9623
20955  9623
20956  9623  b9 00 01  lsg   lda fbuffr,y
20957  9626  c9 2d          cmp #'-'
20958  9628  d0 03          bne nomn
20959  962a
20960  962a  6e 28 01       ror usgn        PRINT USING : Vorzeichen Exponent
20961  962d  c8        nomn  iny
20962  962e  8c 29 01       sty uexp        PRINT USING : Zeiger auf Exponent
20963  9631
20964  9631  a5 80     eoa   lda parsts      DOS Syntax Byte 1
20965  9633  10 02          bpl rtts
20966  9635
20967  9635  86 80          stx parsts      DOS Syntax Byte 1
20968  9637
20969  9637  20 f2 98  rtts  jsr anaf
20970  963a
20971  963a  ad 2c 01       lda vf          PRINT USING : # Position vor
                                             Dezimalpunkt
20972  963d  c9 ff          cmp #$ff
20973  963f  f0 29          beq erst
20974  9641
20975  9641  ad 2f 01       lda fesp        PRINT USING : Exponent-Flag (Feld)
20976  9644  f0 3f          beq cff
20977  9646
20978  9646  ad 29 01       lda uexp        PRINT USING : Zeiger auf Exponent
20979  9649  d0 12          bne ete
20980  964b
20981  964b  ae 24 01       ldx enr         PRINT USING : Zeiger auf Ende
20982  964e  20 74 97       jsr et2
20983  9651  de 02 01       dec fbuffr+2,x
20984  9654  e8             inx
20985  9655  8e 29 01       stx uexp        PRINT USING : Zeiger auf Exponent
20986  9658
20987  9658  20 fb 97       jsr alg
20988  965b  f0 25          beq hup
20989  965d
20990  965d  ac 2e 01  ete   ldy posp        PRINT USING : +/- Flag
20991  9660  d0 17          bne sswe
20992  9662
20993  9662  ac 32 01       ldy sno         PRINT USING : Vorzeichennummer
20994  9665  30 12          bmi sswe
20995  9667
20996  9667  ad 2c 01       lda vf          PRINT USING : # Position vor
                                             Dezimalpunkt
20997  966a  f0 6a     erst  beq errf
20998  966c
20999  966c  ce 2c 01       dec vf          PRINT USING : # Position vor
                                             Dezimalpunkt
21000  966f  d0 05          bne rspa
21001  9671
```

```
21002  9671  ad 2d 01        lda nf          PRINT USING : # Position nach
                                             Dezimalpunkt
21003  9674  f0 60           beq errf
21004  9676
21005  9676  ee 27 01  rspa  inc swe         PRINT USING : Zähler
21006  9679  20 ee 96  sswe  jsr shpn
21007  967c  20 b9 97        jsr uround
21008  967f  20 ee 96        jsr shpn
21009  9682  4c 1c 98  hup   jmp chout
21010  9685
21011  9685
21012  9685  ac 29 01  cff   ldy uexp         PRINT USING : Zeiger auf Exponent
21013  9688  f0 16           beq ftf
21014  968a
21015  968a  85 78           sta hulp        PRINT USING : Zähler
21016  968c
21017  968c  38              sec
21018  968d  6e 30 01        ror etof        PRINT USING : Schalter
21019  9690  a4 80           ldy parsts      DOS Syntax Byte 1
21020  9692  ad 28 01        lda usgn        PRINT USING : Vorzeichen Exponent
21021  9695  10 06           bpl no4
21022  9697
21023  9697  20 27 97        jsr nos3
21024  969a  4c a9 96        jmp rndd
21025  969d
21026  969d
21027  969d  20 08 97  no4   jsr nos4
21028  96a0  a4 80     ftf   ldy parsts      DOS Syntax Byte 1
21029  96a2  f0 05           beq rndd
21030  96a4
21031  96a4  20 ff 97        jsr cho
21032  96a7  f0 06           beq devn
21033  96a9
21034  96a9  20 b9 97  rndd  jsr uround
21035  96ac  4c b2 96        jmp devn2
21036  96af
21037  96af
21038  96af  ce 2a 01  devn  dec vn          PRINT USING : Anzahl Stellen vor
                                             Dezimalpunkt
21039  96b2  38        devn2 sec
21040  96b3  ad 2c 01        lda vf          PRINT USING : # Position vor
                                             Dezimalpunkt
21041  96b6  ed 2a 01        sbc vn          PRINT USING : Anzahl Stellen vor
                                             Dezimalpunkt
21042  96b9  90 1b           bcc errf
21043  96bb
21044  96bb  8d 27 01        sta swe         PRINT USING : Zähler
21045  96be
21046  96be  ac 2e 01        ldy posp        PRINT USING : +/- Flag
21047  96c1  d0 1b           bne ahp
21048  96c3
21049  96c3  ac 32 01        ldy sno         PRINT USING : Vorzeichennummer
21050  96c6  30 16           bmi ahp
21051  96c8
21052  96c8  a8              tay
21053  96c9  f0 0b           beq errf
21054  96cb
21055  96cb  88              dey
```

```
21056  96cc  d0 13           bne ldvn
21057  96ce
21058  96ce  ad 2d 01        lda nf          PRINT USING : # Position nach
                                             Dezimalpunkt
21059  96d1  0d 2a 01        ora vn          PRINT USING : Anzahl Stellen vor
                                             Dezimalpunkt
21060  96d4  d0 ac           bne hup
21061  96d6
21062  96d6  a9 2a     errf  lda #'*'
21063  96d8  20 eb 98  stout jsr cdout
21064  96db  d0 fb           bne stout
21065  96dd
21066  96dd  60               rts
21067  96de
21068  96de
21069  96de  a8         ahp   tay
21070  96df  f0 a1           beq hup
21071  96e1
21072  96e1  ad 2a 01   ldvn  lda vn          PRINT USING : Anzahl Stellen vor
                                             Dezimalpunkt
21073  96e4  d0 9c           bne hup
21074  96e6
21075  96e6  ce 27 01        dec swe         PRINT USING : Zähler
21076  96e9  e6 77           inc mltus1
21077  96eb  4c 82 96        jmp hup
21078  96ee
21079  96ee
21080  96ee  38         shpn  sec
21081  96ef  ad 2c 01        lda vf          PRINT USING : # Position vor
                                             Dezimalpunkt
21082  96f2  ed 2a 01        sbc vn          PRINT USING : Anzahl Stellen vor
                                             Dezimalpunkt
21083  96f5  f0 39           beq rdy
21084  96f7
21085  96f7  a4 80           ldy parsts      DOS Syntax Byte 1
21086  96f9  90 16           bcc pntl
21087  96fb
21088  96fb  85 78           sta hulp        PRINT USING : Zähler
21089  96fd
21090  96fd  cc 24 01   incy  cpy enr         PRINT USING : Zeiger auf Ende
21091  9700  f0 02           beq ity
21092  9702
21093  9702  b0 01           bcs nos1
21094  9704
21095  9704  c8         ity   iny
21096  9705  ee 2a 01   nos1  inc vn          PRINT USING : Anzahl Stellen vor
                                             Dezimalpunkt
21097  9708  20 3d 97   nos4  jsr eado
21098  970b  c6 78           dec hulp        PRINT USING : Zähler
21099  970d  d0 ee           bne incy
21100  970f
21101  970f  f0 1d           beq poit
21102  9711
21103  9711  49 ff      pntl  eor #$ff
21104  9713  69 01           adc #1
21105  9715  85 78           sta hulp        PRINT USING : Zähler
21106  9717
21107  9717  cc 23 01   decy  cpy bnr         PRINT USING : Zeiger auf Anfang
```

```
21108  971a  f0 07          beq inz
21109  971c
21110  971c  88             dey
21111  971d  ce 2a 01       dec vn          PRINT USING : Anzahl Stellen vor
                                             Dezimalpunkt
21112  9720  4c 25 97       jmp inz2
21113  9723
21114  9723
21115  9723  e6 77       inz   inc mltus1
21116  9725  a9 80       inz2  lda #$80
21117  9727  20 3f 97    nos3  jsr eadj
21118  972a  c6 78             dec hulp        PRINT USING : Zähler
21119  972c  d0 e9             bne decy
21120  972e
21121  972e  84 80       poit  sty parsts      DOS Syntax Byte 1
21122  9730
21123  9730  60          rdy   rts
21124  9731
21125  9731
21126  9731  d0 39       sexp  bne retrn
21127  9733
21128  9733  49 09             eor #$09
21129  9735  9d 00 01          sta fbuffr,x
21130  9738
21131  9738  ca                 dex
21132  9739  ec 29 01          cpx uexp        PRINT USING : Zeiger auf Exponent
21133  973c  60                 rts
21134  973d
21135  973d
21136  973d  a9 00       eado  lda #0
21137  973f  ae 29 01    eadj  ldx uexp        PRINT USING : Zeiger auf Exponent
21138  9742  e8                 inx
21139  9743
21140  9743  2c 30 01          bit etof        PRINT USING : Schalter
21141  9746  30 10             bmi tag2
21142  9748
21143  9748  4d 28 01          eor usgn        PRINT USING : Vorzeichen Exponent
21144  974b  f0 0b             beq tag2
21145  974d
21146  974d  20 82 97    tag1  jsr tag3
21147  9750  20 31 97          jsr sexp
21148  9753  b0 f8             bcs tag1
21149  9755
21150  9755  4c 5d 89          jmp overr       Behandlung »?overflow error«
21151  9758
21152  9758
21153  9758  bd 00 01    tag2  lda fbuffr,x
21154  975b  de 00 01          dec fbuffr,x
21155  975e  c9 30             cmp #'0'
21156  9760  20 31 97          jsr sexp
21157  9763  b0 f3             bcs tag2
21158  9765
21159  9765  2c 30 01          bit etof        PRINT USING : Schalter
21160  9768  10 05             bpl et3
21161  976a
21162  976a  84 80             sty parsts      DOS Syntax Byte 1
21163  976c
21164  976c  68          retrn pla
```

```
21165  976d  68              pla
21166  976e  60              rts
21167  976f
21168  976f
21169  976f  ad 28 01   et3   lda usgn        PRINT USING : Vorzeichen Exponent
21170  9772  49 80            eor #$80
21171  9774  8d 28 01   et2   sta usgn        PRINT USING : Vorzeichen Exponent
21172  9777
21173  9777  a9 30            lda #'0'
21174  9779  9d 01 01         sta fbuffr+1,x
21175  977c
21176  977c  a9 31            lda #'1'
21177  977e  9d 02 01         sta fbuffr+2,x
21178  9781
21179  9781  60              rts
21180  9782
21181  9782
21182  9782  bd 00 01   tag3  lda fbuffr,x
21183  9785  fe 00 01         inc fbuffr,x
21184  9788  c9 39            cmp #'9'
21185  978a  60              rts
21186  978b
21187  978b
21188  978b  18        ansub clc
21189  978c  c8              iny
21190  978d  f0 05           beq ans010
21191  978f
21192  978f  cc 35 01        cpy lfor          PRINT USING : Länge Formatstring
21193  9792  90 04           bcc ans020
21194  9794
21195  9794  a4 77    ans010 ldy mltus1
21196  9796  d0 d4           bne retrn
21197  9798
21198  9798  20 d3 42 ans020 jsr indfrm       LDA (fndpnt),y aus VAR-RAM
21199  979b  ee 31 01         inc cform         PRINT USING : Zeichen-Zähler
21200  979e  60              rts
21201  979f
21202  979f
21203  979f  20 81 87   ini   jsr frefac       Daten Arbeitsstring aus Ablage,
                                               Adresse index1, Länge ac
21204  97a2  85 78           sta hulp          PRINT USING : Zähler
21205  97a4
21206  97a4  a2 0a           ldx #10
21207  97a6  a9 00           lda #0
21208  97a8  9d 27 01   stzl  sta swe,x         PRINT USING : Zähler
21209  97ab  ca              dex
21210  97ac  10 fa           bpl stzl
21211  97ae
21212  97ae  8e 26 01        stx flag          PRINT USING : Komma-Flag
21213  97b1  86 80           stx parsts        DOS Syntax Byte 1
21214  97b3  8e 25 01        stx dolr          PRINT USING : Dollar-Flag
21215  97b6
21216  97b6  aa              tax
21217  97b7  a8              tay
21218  97b8
21219  97b8  60              rts
21220  97b9
21221  97b9
```

```
21222  97b9  18          uround clc
21223  97ba  a5 80              lda parsts       DOS Syntax Byte 1
21224  97bc  6d 2d 01           adc nf           PRINT USING : # Position nach
                                                 Dezimalpunkt
21225  97bf  b0 39              bcs rrts
21226  97c1
21227  97c1  38                 sec
21228  97c2  e5 77              sbc mltus1
21229  97c4  90 34              bcc rrts
21230  97c6
21231  97c6  cd 24 01           cmp enr          PRINT USING : Zeiger auf Ende
21232  97c9  f0 02              beq cbn
21233  97cb
21234  97cb  b0 2d              bcs rrts
21235  97cd
21236  97cd  cd 23 01    cbn    cmp bnr          PRINT USING : Zeiger auf Anfang
21237  97d0  90 28              bcc rrts
21238  97d2
21239  97d2  aa                 tax
21240  97d3  bd 00 01           lda fbuffr,x
21241  97d6  c9 35              cmp #'5'
21242  97d8  90 20              bcc rrts
21243  97da
21244  97da  ec 23 01    con1   cpx bnr          PRINT USING : Zeiger auf Anfang
21245  97dd  f0 0a              beq add1
21246  97df
21247  97df  ca                 dex
21248  97e0  20 82 97           jsr tag3
21249  97e3  8e 24 01           stx enr          PRINT USING : Zeiger auf Ende
21250  97e6
21251  97e6  f0 f2              beq con1
21252  97e8
21253  97e8  60                 rts
21254  97e9
21255  97e9
21256  97e9  a9 31       add1   lda #'1'
21257  97eb  9d 00 01           sta fbuffr,x
21258  97ee
21259  97ee  e8                 inx
21260  97ef  86 80              stx parsts       DOS Syntax Byte 1
21261  97f1
21262  97f1  c6 77              dec mltus1
21263  97f3  10 05              bpl rrts
21264  97f5
21265  97f5  e6 77              inc mltus1
21266  97f7  ee 2a 01           inc vn           PRINT USING : Anzahl Stellen vor
                                                 Dezimalpunkt
21267  97fa  60          rrts   rts
21268  97fb
21269  97fb
21270  97fb  a4 80       alg    ldy parsts       DOS Syntax Byte 1
21271  97fd  f0 17              beq szer
21272  97ff
21273  97ff  ac 23 01    cho    ldy bnr          PRINT USING : Zeiger auf Anfang
21274  9802  b9 00 01    cmo    lda fbuffr,y
21275  9805  c9 30              cmp #'0'
21276  9807  60                 rts
21277  9808
```

```
21278  9808
21279  9808  e6 80       nbr    inc parsts      DOS Syntax Byte 1
21280  980a  20 3d 97            jsr eado
21281  980d  ee 23 01            inc bnr         PRINT USING : Zeiger auf Anfang
21282  9810  cc 24 01            cpy enr         PRINT USING : Zeiger auf Ende
21283  9813  f0 e5               beq rrts
21284  9815
21285  9815  c8                  iny
21286  9816  20 02 98    szer    jsr cmo
21287  9819  f0 ed               beq nbr
21288  981b
21289  981b  60                  rts
21290  981c
21291  981c
21292  981c  ad 25 01    chout   lda dolr        PRINT USING : Dollar-Flag
21293  981f  30 02               bmi chou
21294  9821
21295  9821  e6 77               inc mltus1
21296  9823  ae 23 01    chou    ldx bnr         PRINT USING : Zeiger auf Anfang
21297  9826  ca                  dex
21298  9827  ac 34 01            ldy begfd       PRINT USING : Zeiger auf Feld
21299  982a  20 d3 42    afrm    jsr indfrm      LDA (fndpnt),y aus VAR-RAM
21300  982d  c8                  iny
21301  982e  c9 2c               cmp #','
21302  9830  d0 14               bne punt
21303  9832
21304  9832  2c 26 01            bit flag        PRINT USING : Komma-Flag
21305  9835  30 09               bmi bout
21306  9837
21307  9837  8d 03 ff            sta lcrc        ROM / PGM-RAM einschalten
21308  983a
21309  983a  ad 05 12            lda pucoma
21310  983d  4c ab 98            jmp out
21311  9840
21312  9840
21313  9840  ad 33 01    bout    lda blfd        PRINT USING : Blank/Stern-Flag
21314  9843  4c ab 98            jmp out
21315  9846
21316  9846
21317  9846  c9 2e       punt    cmp #'.'
21318  9848  d0 09               bne afplus
21319  984a
21320  984a  8d 03 ff            sta lcrc        ROM / PGM-RAM einschalten
21321  984d
21322  984d  ad 06 12            lda pudot
21323  9850  4c ab 98            jmp out
21324  9853
21325  9853
21326  9853  c9 2b       afplus  cmp #'+'
21327  9855  f0 3b               beq ispl
21328  9857
21329  9857  c9 2d               cmp #'-'
21330  9859  f0 32               beq ispl1
21331  985b
21332  985b  c9 5e               cmp #$5e        Potenz
21333  985d  d0 69               bne pndd
21334  985f
21335  985f  a9 45               lda #'e'
```

```
21336  9861  20 eb 98        jsr cdout
21337  9864  ac 29 01        ldy uexp        PRINT USING : Zeiger auf Exponent
21338  9867  20 02 98        jsr cmo
21339  986a  d0 06           bne mint
21340  986c
21341  986c  c8              iny
21342  986d  20 02 98        jsr cmo
21343  9870  f0 07           beq post
21344  9872
21345  9872  a9 2d     mint   lda #'-'
21346  9874  2c 28 01        bit usgn        PRINT USING : Vorzeichen Exponent
21347  9877  30 02           bmi mout
21348  9879
21349  9879  a9 2b     post   lda #'+'
21350  987b  20 eb 98   mout   jsr cdout
21351  987e
21352  987e  ae 29 01        ldx uexp        PRINT USING : Zeiger auf Exponent
21353  9881  bd 00 01        lda fbuffr,x
21354  9884  20 eb 98        jsr cdout
21355  9887
21356  9887  ac 36 01        ldy endfd
21357  988a  4c a1 98        jmp telx
21358  988d
21359  988d
21360  988d  ad 32 01   ispl1 lda sno         PRINT USING : Vorzeichennummer
21361  9890  30 ae           bmi bout
21362  9892
21363  9892  ad 32 01   ispl  lda sno         PRINT USING : Vorzeichennummer
21364  9895  4c ab 98        jmp out
21365  9898
21366  9898
21367  9898  a5 77     zerot  lda mltus1
21368  989a  d0 18           bne zerot1
21369  989c
21370  989c  ec 24 01        cpx enr         PRINT USING : Zeiger auf Ende
21371  989f  f0 05           beq zout
21372  98a1
21373  98a1  e8        telx   inx
21374  98a2  bd 00 01        lda fbuffr,x
21375  98a5  2c              .byte _skip2
21376  98a6
21377  98a6  a9 30     zout   lda #'0'
21378  98a8  4e 26 01  outs   lsr flag        PRINT USING : Komma-Flag
21379  98ab  20 eb 98  out    jsr cdout
21380  98ae  f0 03           beq outrt
21381  98b0
21382  98b0  4c 2a 98        jmp afrm
21383  98b3
21384  98b3
21385  98b3  60        outrt  rts
21386  98b4
21387  98b4
21388  98b4  c6 77     zerot1 dec mltus1
21389  98b6  ad 25 01        lda dolr        PRINT USING : Dollar-Flag
21390  98b9  30 eb           bmi zout
21391  98bb
21392  98bb  38              sec
21393  98bc  6e 25 01        ror dolr        PRINT USING : Dollar-Flag
```

```
21394  98bf  8d 03 ff          sta lcrc        ROM / PGM-RAM einschalten
21395  98c2
21396  98c2  ad 07 12          lda pumony
21397  98c5  4c a8 98          jmp outs
21398  98c8
21399  98c8
21400  98c8  ad 27 01   pndd    lda swe         PRINT USING : Zähler
21401  98cb  f0 cb              beq zerot
21402  98cd
21403  98cd  ce 27 01           dec swe         PRINT USING : Zähler
21404  98d0  f0 03      pndx    beq pndy
21405  98d2
21406  98d2  4c 40 98           jmp bout
21407  98d5
21408  98d5
21409  98d5  ad 2e 01   pndy    lda posp        PRINT USING : +/- Flag
21410  98d8  30 f6              bmi pndx
21411  98da
21412  98da  20 d3 42   tat     jsr indfrm      LDA (fndpnt),y aus VAR-RAM
21413  98dd  c9 2c              cmp #','
21414  98df  d0 ac              bne ispl1
21415  98e1
21416  98e1  ad 33 01           lda blfd        PRINT USING : Blank/Stern-Flag
21417  98e4  20 eb 98           jsr cdout
21418  98e7  c8                 iny
21419  98e8  4c da 98           jmp tat
21420  98eb
21421  98eb
21422  98eb  20 0c 56   cdout   jsr outdo       Ausgabe ac auf akt. Kanal
21423  98ee  ce 31 01           dec cform       PRINT USING : Zeichen-Zähler
21424  98f1  60                 rts
21425  98f2
21426  98f2
21427  98f2  ac 36 01   anaf    ldy endfd
21428  98f5  20 8b 97   gfor    jsr ansub
21429  98f8  20 a7 99           jsr comp
21430  98fb  d0 14              bne pchar
21431  98fd
21432  98fd  8c 34 01           sty begfd       PRINT USING : Zeiger auf Feld
21433  9900  90 1a              bcc ffoun
21434  9902
21435  9902  aa                 tax
21436  9903  20 8b 97   sfur    jsr ansub
21437  9906  b0 05              bcs nono
21438  9908
21439  9908  20 af 99           jsr com1
21440  990b  f0 0a              beq foun1
21441  990d
21442  990d  ac 34 01   nono    ldy begfd       PRINT USING : Zeiger auf Feld
21443  9910  8a                 txa
21444  9911  20 0c 56   pchar   jsr outdo       Ausgabe ac auf akt. Kanal
21445  9914  4c f5 98           jmp gfor
21446  9917
21447  9917
21448  9917  b0 ea      foun1   bcs sfur
21449  9919
21450  9919  ac 34 01           ldy begfd       PRINT USING : Zeiger auf Feld
21451  991c  a6 77      ffoun   ldx mltus1
```

```
21452  991e  d0 7a          bne trts
21453  9920
21454  9920  8e 31 01       stx cform       PRINT USING : Zeichen-Zähler
21455  9923
21456  9923  88             dey
21457  9924  ce 31 01  hyo2  dec cform       PRINT USING : Zeichen-Zähler
21458  9927  20 8b 97  hyo   jsr ansub
21459  992a  b0 74          bcs efo
21460  992c
21461  992c  c9 2c          cmp #','
21462  992e  f0 f7          beq hyo
21463  9930
21464  9930  20 7e 99       jsr isp
21465  9933  90 ef          bcc hyo2
21466  9935
21467  9935  c9 2e          cmp #'.'
21468  9937  d0 08          bne avf1
21469  9939
21470  9939  e8             inx
21471  993a  e0 02          cpx #2
21472  993c  90 e9          bcc hyo
21473  993e
21474  993e  4c 6c 79  ero   jmp snerr       Behandlung »?syntax error«
21475  9941
21476  9941
21477  9941  20 b3 99  avf1  jsr com2
21478  9944  d0 0b          bne llar
21479  9946  90 03          bcc hyo1
21480  9948
21481  9948  8d 2b 01       sta chsn        PRINT USING : justify Flag
21482  994b
21483  994b  fe 2c 01  hyo1  inc vf,x        PRINT USING : # Position vor
                                             Dezimalpunkt
21484  994e  4c 27 99       jmp hyo
21485  9951
21486  9951
21487  9951  c9 24      llar  cmp #'$'
21488  9953  d0 0f          bne expo
21489  9955
21490  9955  2c 25 01       bit dolr        PRINT USING : Dollar-Flag
21491  9958  10 f1          bpl hyo1
21492  995a
21493  995a  18             clc
21494  995b  6e 25 01       ror dolr        PRINT USING : Dollar-Flag
21495  995e  ce 2c 01       dec vf          PRINT USING : # Position vor
                                             Dezimalpunkt
21496  9961  4c 4b 99       jmp hyo1
21497  9964
21498  9964
21499  9964  c9 5e      expo  cmp #$5e        Potenz
21500  9966  d0 16          bne isp
21501  9968
21502  9968  a2 02          ldx #2
21503  996a  20 8b 97  chkk  jsr ansub
21504  996d  b0 cf          bcs ero
21505  996f
21506  996f  c9 5e          cmp #$5e        Potenz
21507  9971  d0 cb          bne ero
```

```
21508  9973
21509  9973  ca              dex
21510  9974  10 f4           bpl chkk
21511  9976
21512  9976  ee 2f 01        inc fesp        PRINT USING : Exponent-Flag (Feld)
21513  9979  20 8b 97        jsr ansub
21514  997c  b0 22           bcs efo
21515  997e
21516  997e  c9 2b     isp    cmp #'+'
21517  9980  d0 19           bne chom
21518  9982
21519  9982  ad 32 01        lda sno         PRINT USING : Vorzeichennummer
21520  9985  10 05           bpl spos
21521  9987
21522  9987  a9 2b           lda #'+'
21523  9989  8d 32 01        sta sno         PRINT USING : Vorzeichennummer
21524  998c
21525  998c  ad 2e 01  spos   lda posp        PRINT USING : +/- Flag
21526  998f  d0 ad           bne ero
21527  9991
21528  9991  6e 2e 01        ror posp        PRINT USING : +/- Flag
21529  9994  8c 36 01        sty endfd
21530  9997
21531  9997  ee 31 01        inc cform       PRINT USING : Zeichen-Zähler
21532  999a  60        trts   rts
21533  999b
21534  999b
21535  999b  c9 2d     chom   cmp #'-'
21536  999d  f0 ed           beq spos
21537  999f
21538  999f  38              sec
21539  99a0  8c 36 01  efo    sty endfd
21540  99a3
21541  99a3  ce 36 01        dec endfd
21542  99a6  60              rts
21543  99a7
21544  99a7
21545  99a7  c9 2b     comp   cmp #'+'
21546  99a9  f0 15           beq rt
21547  99ab
21548  99ab  c9 2d           cmp #'-'
21549  99ad  f0 11           beq rt
21550  99af
21551  99af  c9 2e     com1   cmp #'.'
21552  99b1  f0 0d           beq rt
21553  99b3
21554  99b3  c9 3d     com2   cmp #'='
21555  99b5  f0 09           beq rt
21556  99b7
21557  99b7  c9 3e           cmp #$3e
21558  99b9  f0 05           beq rt
21559  99bb
21560  99bb  c9 23           cmp #'#'
21561  99bd  d0 01           bne rt
21562  99bf
21563  99bf  18              clc
21564  99c0  60        rt     rts
21565  99c1
```

```
21566  99c1
21567  99c1                    .end
21568  99c1                    .lib basic.instr
21569  99c1  put"@basic.instr"
```

```
21571  99c1
21572  99c1
21573  99c1  ===> BASIC-Funktion INSTR ($d4) <===
21574  99c1
21575  99c1  a5 66      instr  lda indice       ADDRL Descr. Durchsuch String
21576  99c3  8d d6 03         sta tmdes1       ADDRL Desc.
21577  99c6
21578  99c6  a5 67            lda indice+1     ADDRH Descr. Durchsuch String
21579  99c8  8d d7 03         sta tmdes1+1     ADDRH Desc.
21580  99cb
21581  99cb  ===> Param. Suchstring einlesen <===
21582  99cb
21583  99cb  20 ef 77         jsr frmevl       folgenden Ausdruck auswerten
21584  99ce  20 dd 77         jsr chkstr       Test String VAR, sonst type mism.
21585  99d1
21586  99d1  a5 66            lda indice       ADDRL Descr. Suchstring
21587  99d3  8d d8 03         sta tmdes2       ADDRL Desc.
21588  99d6
21589  99d6  a5 67            lda indice+1     ADDRH Descr. Suchstring
21590  99d8  8d d9 03         sta tmdes2+1     ADDRH Desc.
21591  99db
21592  99db  a2 01            ldx #1           Vorgabe Startspalte Suchen
21593  99dd  86 67            stx faclo        Startspalte Suchen
21594  99df
21595  99df  20 86 03         jsr chrgot       letztes Zeichen erneut nach ac
21596  99e2  c9 29            cmp #')'
21597  99e4  f0 03            beq inst1
21598  99e6
21599  99e6  ===> Startspalte Suchen einlesen <===
21600  99e6
21601  99e6  20 09 88         jsr combyt       Kommatest, 8 Bit INT xr/faclo
21602  99e9
21603  99e9  20 56 79   inst1 jsr chkcls       Test auf folg. »)«, sonst Fehler
21604  99ec
21605  99ec  a6 67            ldx faclo        Startspalte Suchen
21606  99ee  d0 03            bne inst2
21607  99f0
21608  99f0  ===> Behandlung »?illegal quantity error« <===
21609  99f0
21610  99f0  4c 28 7d         jmp fcerr        Behandlung »?illegal quantity error«
21611  99f3
21612  99f3
21613  99f3  ca         inst2 dex              Zeiger in Durchsuch String
21614  99f4  86 63            stx positn       Zeiger in Durchsuch String
21615  99f6
21616  99f6  ===> Zeiger auf Descr. nach Zeropage <===
21617  99f6
21618  99f6  a2 03            ldx #3
21619  99f8  bd d6 03   inst3 lda tmdes1,x     Zeiger auf Descr. String 1/2
21620  99fb  95 59            sta ptarg1,x
21621  99fd  ca               dex
21622  99fe  10 f8            bpl inst3
21623  9a00
21624  9a00  ===> Descr. Durchsuch String nach Zeropage <===
21625  9a00
21626  9a00  a0 02            ldy #2
21627  9a02  a9 59      inst4 lda #<ptarg1     Zeiger Descr. Durchsuch String
21628  9a04  20 ab 03         jsr iram1        LDA (ac),y aus VAR-RAM
```

```
21629  9a07  99 5d 00          sta str1,y       Descr. Durchsuch String
21630  9a0a
21631  9a0a  ===> Descr. Suchstring nach Zeropage <===
21632  9a0a
21633  9a0a  a9 5b             lda #<ptarg2     Zeiger Descr. Suchstring
21634  9a0c  20 ab 03          jsr iram1        LDA (ac),y aus VAR-RAM
21635  9a0f  99 60 00          sta str2,y       Descr. Suchstring
21636  9a12
21637  9a12  88                dey
21638  9a13  10 ed             bpl inst4
21639  9a15
21640  9a15  ===> Test Suchstring leer <===
21641  9a15
21642  9a15  a5 60             lda str2         Länge Suchstring
21643  9a17  f0 3b             beq instnf       Übergabe Wert 0
21644  9a19
21645  9a19
21646  9a19  ===> ab nächster Spalte suchen <===
21647  9a19
21648  9a19  a9 00       inst5 lda #0
21649  9a1b  85 64             sta match        Zeiger in Suchstring
21650  9a1d
21651  9a1d  ===> Test Rest Durchsuchstr. < Suchstring <===
21652  9a1d
21653  9a1d  18                clc
21654  9a1e  a5 60             lda str2         Länge Suchstring
21655  9a20  65 63             adc positn       Zeiger in Durchsuch String
21656  9a22  b0 30             bcs instnf       Übergabe Wert 0
21657  9a24
21658  9a24  c5 5d             cmp str1         Länge Durchsuch String
21659  9a26  90 02             bcc inst6        nächstes Zeichen vergleichen
21660  9a28  d0 2a             bne instnf       Übergabe Wert 0
21661  9a2a
21662  9a2a
21663  9a2a  ===> nächstes Zeichen vergleichen <===
21664  9a2a
21665  9a2a  a4 64       inst6 ldy match        Zeiger in Suchstring
21666  9a2c  c4 60             cpy str2         Länge Suchstring
21667  9a2e  f0 1f             beq instfd       Suchstring gefunden, Übergabe Spalte
21668  9a30
21669  9a30  98                tya              Zeiger in Suchstring
21670  9a31  18                clc
21671  9a32  65 63             adc positn       Zeiger in Durchsuch String
21672  9a34
21673  9a34  ===> Zeichen aus Durchsuch String laden <===
21674  9a34
21675  9a34  a8                tay              neuer Zeiger in Durchsuch String
21676  9a35  a9 5e             lda #<str1+1     ADDRL Durchsuch String
21677  9a37  20 ab 03          jsr iram1        LDA (ac),y aus VAR-RAM
21678  9a3a  85 79             sta syntmp       Zeichen aus Durchsuch String
21679  9a3c
21680  9a3c  ===> mit Zeichen aus Suchstring vergleichen <===
21681  9a3c
21682  9a3c  a4 64             ldy match        Zeiger in Suchstring
21683  9a3e  a9 61             lda #<str2+1     ADDRL Suchstring
21684  9a40  20 ab 03          jsr iram1        LDA (ac),y aus VAR-RAM
21685  9a43
21686  9a43  c5 79             cmp syntmp       Zeichen aus Durchsuch String
```

```
21687  9a45  f0 04            beq inst7        Zeiger Suchstring erhöhen
21688  9a47
21689  9a47  e6 63            inc positn       Zeiger in Durchsuch String
21690  9a49  d0 ce            bne inst5        ab nächster Spalte suchen
21691  9a4b
21692  9a4b
21693  9a4b  ===> Zeiger Suchstring erhöhen <===
21694  9a4b
21695  9a4b  e6 64      inst7 inc match        Zeiger in Suchstring
21696  9a4d  d0 db            bne inst6        nächstes Zeichen vergleichen
21697  9a4f
21698  9a4f
21699  9a4f  ===> Suchstring gefunden, Übergabe Spalte <===
21700  9a4f
21701  9a4f  e6 63      instfd inc positn       Zeiger in Durchsuch String
21702  9a51  a5 63            lda positn       Spalte in Durchsuch String
21703  9a53  2c              .byte _skip2
21704  9a54
21705  9a54
21706  9a54  ===> Übergabe Wert 0 <===
21707  9a54
21708  9a54  a9 00      instnf lda #0
21709  9a56  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
21710  9a59  48                pha              Übergabe Wert
21711  9a5a
21712  9a5a  ad d8 03          lda tmdes2       ADDRL Descr. Suchstring
21713  9a5d  ac d9 03          ldy tmdes2+1     ADDRH Descr. Suchstring
21714  9a60  20 85 87          jsr fretmp       String ac/yr vom Stapel, ungültig
21715  9a63  8d 03 ff          sta lcrc         ROM / PGM-RAM einschalten
21716  9a66
21717  9a66  ad d6 03          lda tmdes1       ADDRL Descr. Durchsuch String
21718  9a69  ac d7 03          ldy tmdes1+1     ADDRH Descr. Durchsuch String
21719  9a6c  20 85 87          jsr fretmp       String ac/yr vom Stapel, ungültig
21720  9a6f
21721  9a6f  68                pla              Übergabe Wert (Spalte oder 0)
21722  9a70  a8                tay
21723  9a71  4c d4 84          jmp sngflt       8 Bit INT yr nach GK in FAC
21724  9a74
21725  9a74
21726  9a74                    .end
21727  9a74                    .lib basic.graph_4
21728  9a74  put"@basic.graph_4"
```

```
21730  9a74
21731  9a74
21732  9a74  ===> SIN/COS Winkel (yr) nach sinpc/cospc <===
21733  9a74
21734  9a74  20 8f 9d   getand jsr settwo       Winkel (yr) nach ac/yr
21735  9a77
21736  9a77
21737  9a77  ===> SIN/COS ac/yr nach sinpc/cospc <===
21738  9a77
21739  9a77  a2 00      gtang1 ldx #0           Quadrant
21740  9a79
21741  9a79
21742  9a79  ===> Winkel ac/yr durch 90 divid. <===
21743  9a79
21744  9a79  e8         getan1 inx              Quadrant zählen
21745  9a7a  38                sec
21746  9a7b  e9 5a             sbc #90
21747  9a7d  b0 fa             bcs getan1      Winkel ac/yr durch 90 divid.
21748  9a7f
21749  9a7f  88                dey             High-Byte Koord.
21750  9a80  10 f7             bpl getan1      Winkel ac/yr durch 90 divid.
21751  9a82
21752  9a82  8e 49 11          stx angsgn      Quadrant
21753  9a85
21754  9a85  48                pha             neg. Winkel für COS Berechnung
21755  9a86
21756  9a86  69 5a             adc #90         Winkel für SIN Berechnung
21757  9a88  20 94 9a          jsr getan2      SIN Winkel in ac (0-90) berechnen
21758  9a8b
21759  9a8b  68                pla             neg. Winkel
21760  9a8c  18                clc
21761  9a8d  49 ff             eor #%11111111
21762  9a8f  69 01             adc #1          Winkel pos. machen
21763  9a91  ce 49 11          dec angsgn      Quadrant
21764  9a94
21765  9a94   SIN (90 - Winkel) = COS (Winkel)
21766  9a94
21767  9a94
21768  9a94  ===> SIN Winkel in ac (0-90) berechnen <===
21769  9a94
21770  9a94  a2 ff      getan2 ldx #255
21771  9a96
21772  9a96
21773  9a96  ===> Winkel (0-90) durch 10 divid. <===
21774  9a96
21775  9a96  e8         getan3 inx              Quotient
21776  9a97  38                sec
21777  9a98  e9 0a             sbc #10
21778  9a9a  b0 fa             bcs getan3      Winkel (0-90) durch 10 divid.
21779  9a9c
21780  9a9c  69 0a             adc #10
21781  9a9e  85 8e             sta vtemp1      Wert Einerstelle Winkel
21782  9aa0
21783  9aa0  8a                txa             Quotient
21784  9aa1  0a                asl a           mal zwei
21785  9aa2  aa                tax             Zeiger in SIN Tabelle
21786  9aa3
21787  9aa3  bd 2a 9f          lda angval+1,x  Low-Byte SIN/COS
```

```
21788  9aa6  bc 29 9f          ldy angval,x     High-Byte SIN/COS
21789  9aa9
21790  9aa9
21791  9aa9  ===> Interpol. für Einerstelle Winkel <===
21792  9aa9
21793  9aa9  18         getan4 clc
21794  9aaa  c6 8e             dec vtemp1       Wert Einerstelle Winkel
21795  9aac  30 0c             bmi getan5       SIN/COS nach sinpc/cospc
21796  9aae
21797  9aae  7d 3e 9f          adc incval+1,x   Low-Byte Interpol.
21798  9ab1  48                pha
21799  9ab2
21800  9ab2  98                tya              High-Byte SIN/COS
21801  9ab3  7d 3d 9f          adc incval,x     High-Byte Interpol.
21802  9ab6
21803  9ab6  a8                tay              High-Byte interpol. SIN/COS
21804  9ab7  68                pla              Low-Byte interpol. SIN/COS
21805  9ab8  90 ef             bcc getan4       immer
21806  9aba
21807  9aba
21808  9aba  ===> SIN/COS nach sinpc/cospc <===
21809  9aba
21810  9aba  48         getan5 pha              Low-Byte SIN/COS
21811  9abb
21812  9abb  a2 00             ldx #0           Zeiger für SIN
21813  9abd  ad 49 11          lda angsgn       Quadrant
21814  9ac0  4a                lsr a
21815  9ac1  b0 02             bcs getan6
21816  9ac3
21817  9ac3  a2 02             ldx #2           Zeiger für COS
21818  9ac5  68         getan6 pla              Low-Byte SIN/COS
21819  9ac6  9d 4a 11          sta sinval,x     sinpc/cospc
21820  9ac9
21821  9ac9  98                tya              High-Byte SIN/COS
21822  9aca  9d 4b 11          sta sinval+1,x   sinpc+1/cospc+1
21823  9acd
21824  9acd  60                rts
21825  9ace
21826  9ace
21827  9ace  ===> Koord.(xr) mal SIN/COS nach ac/yr <===
21828  9ace
21829  9ace  a0 19      angmlt ldy #sinval-vwork      Offset SIN
21830  9ad0  90 02             bcc angm10
21831  9ad2
21832  9ad2  a0 1b             ldy #cosval-vwork     Offset COS
21833  9ad4
21834  9ad4  ad 49 11   angm10 lda angsgn       Quadrant
21835  9ad7  69 02             adc #$02
21836  9ad9  4a                lsr a
21837  9ada  4a                lsr a
21838  9adb  08                php
21839  9adc  20 8f 9d          jsr settwo       SIN/COS nach ac/yr
21840  9adf  c0 ff             cpy #$ff
21841  9ae1  90 07             bcc angm20
21842  9ae3
21843  9ae3  8a                txa
21844  9ae4  a8                tay
21845  9ae5  20 8f 9d          jsr settwo       Winkel (yr) nach ac/yr
```

```
21846  9ae8  b0 03            bcs angm30       immer
21847  9aea
21848  9aea
21849  9aea  20 ae 9d   angm20 jsr twobyt      Koord.(xr) mal ac/yr durch 65536
21850  9aed  28         angm30 plp
21851  9aee  b0 1b             bcs angd20
21852  9af0
21853  9af0  4c 9e 9d          jmp invert       neg. Differenz invertieren
21854  9af3
21855  9af3
21856  9af3  ===> Dist.Koord. mit SIN/COS mult. <===
21857  9af3
21858  9af3  8d 4e 11   angdst sta angcnt
21859  9af6
21860  9af6  a2 23             ldx #xdist1-vwork
21861  9af8  0e 4e 11   angd10 asl angcnt
21862  9afb  20 ce 9a          jsr angmlt       Koord.(xr) mal SIN/COS nach ac/yr
21863  9afe  9d 31 11          sta vwork,x
21864  9b01
21865  9b01  98                tya
21866  9b02  9d 32 11          sta xypos+1,x    Ord. High-PIXEL CURS.
21867  9b05
21868  9b05  e8                inx
21869  9b06  e8                inx
21870  9b07  e0 2b             cpx #disend-vwork
21871  9b09  90 ed             bcc angd10
21872  9b0b
21873  9b0b  60         angd20 rts
21874  9b0c
21875  9b0c
21876  9b0c  ===> BASIC-Funktion RDOT ($d0) <===
21877  9b0c
21878  9b0c  20 f7 87   rdot   jsr conint       8 Bit INT CONST aus FAC nach xr
21879  9b0f  e0 02             cpx #2
21880  9b11  90 10             bcc rdot30       Absz. bzw. Ord. akt. Pixel Curs. Pos.
21881  9b13  f0 03             beq rdot10       Farbquell-Code akt. Pixel Curs. Pos.
21882  9b15
21883  9b15  ===> Behandlung »?illegal quantity error« <===
21884  9b15
21885  9b15  4c 28 7d          jmp fcerr        Behandlung »?illegal quantity error«
21886  9b18
21887  9b18
21888  9b18  ===> Farbquell-Code akt. Pixel Curs. Pos. <===
21889  9b18
21890  9b18  20 49 9c   rdot10 jsr readpt       Test auf Punkt xr/yr gesetzt
21891  9b1b  a8                tay
21892  9b1c  90 02             bcc rdot20       yr nach GK in FAC
21893  9b1e
21894  9b1e  ===> Punkt nicht im Graphikbereich <===
21895  9b1e
21896  9b1e  a0 00             ldy #0           Ersatzwert
21897  9b20  4c d4 84   rdot20 jmp sngflt       8 Bit INT yr nach GK in FAC
21898  9b23
21899  9b23
21900  9b23  ===> Absz. bzw. Ord. akt. Pixel Curs. Pos. <===
21901  9b23
21902  9b23  8a         rdot30 txa
21903  9b24  0a                asl a
```

```
21904  9b25  aa                tax
21905  9b26  bd 31 11          lda vwork,x
21906  9b29  a8                tay
21907  9b2a  bd 32 11          lda xypos+1,x     Ord. High-PIXEL CURS.
21908  9b2d  4c 3c 79          jmp givayv        16 Bit INT yr/ac nach GK in FAC
21909  9b30
21910  9b30
21911  9b30  ===> Strecke zeichnen <===
21912  9b30
21913  9b30  a2 02       drawln ldx #ypos-vwork
21914  9b32  a0 06              ldy #ydest-vwork
21915  9b34
21916  9b34  a9 00       drw01  lda #0
21917  9b36  9d 3d 11           sta ysgn-2,x
21918  9b39  9d 3e 11           sta ysgn+1-2,x
21919  9b3c
21920  9b3c  20 99 9d           jsr abstwo        Abs.wert Koord.Diff. (yr,xr) nach
                                                 ac, yr
21921  9b3f  10 08              bpl drw02         Diff. nicht negativ
21922  9b41
21923  9b41  de 3d 11           dec xysgn,x
21924  9b44  de 3e 11           dec xysgn+1,x
21925  9b47  d0 0b              bne drw04         immer
21926  9b49
21927  9b49
21928  9b49  ===> Test Koord.Diff. =0 <===
21929  9b49
21930  9b49  c9 00       drw02  cmp #0
21931  9b4b  d0 04              bne drw03
21932  9b4d  c0 00              cpy #0
21933  9b4f  f0 03              beq drw04
21934  9b51
21935  9b51  fe 3d 11    drw03  inc xysgn,x
21936  9b54  9d 39 11    drw04  sta xyabs,x
21937  9b57
21938  9b57  0a                 asl a
21939  9b58  9d 41 11           sta fct,x
21940  9b5b
21941  9b5b  98                 tya
21942  9b5c  9d 3a 11           sta xyabs+1,x
21943  9b5f
21944  9b5f  2a                 rol a
21945  9b60  9d 42 11           sta fct+1,x
21946  9b63
21947  9b63  ca                 dex
21948  9b64  ca                 dex               Zeiger auf xpos
21949  9b65
21950  9b65  a0 04              ldy #xdest-vwork
21951  9b67  e0 00              cpx #vwork-vwork
21952  9b69  f0 c9              beq drw01         nochmal für Abszisse
21953  9b6b
21954  9b6b  a2 0a              ldx #yabs-vwork
21955  9b6d  a0 08              ldy #xabs-vwork
21956  9b6f  20 7c 9d           jsr subtwo        Koord.(yr) minus Koord.(xr) nach
                                                 ac/yr
21957  9b72   nicht neg. Diff.: CARRY = 1
21958  9b72
21959  9b72
```

```
21959  9b72  ===> Berechnung der Zeiger <===
21960  9b72
21961  9b72  a9 00           lda #0
21962  9b74  2a              rol a
21963  9b75  2a              rol a
21964  9b76  8d 47 11        sta lesser      0, wenn negativ, sonst 2
21965  9b79
21966  9b79  49 02           eor #2
21967  9b7b  8d 48 11        sta greatr      2, wenn negativ, sonst 0
21968  9b7e
21969  9b7e  18              clc
21970  9b7f  a9 10           lda #16
21971  9b81  6d 47 11        adc lesser
21972  9b84  a8              tay             16, wenn negativ, sonst 18
21973  9b85  48              pha
21974  9b86
21975  9b86  49 02           eor #2
21976  9b88  aa              tax             18, wenn negativ, sonst 16
21977  9b89
21978  9b89  20 7c 9d        jsr subtwo      Koord.(yr) minus Koord.(xr) nach
                                             ac/yr
21979  9b8c
21980  9b8c  9d 31 11        sta vwork,x
21981  9b8f  98              tya
21982  9b90  9d 32 11        sta xypos+1,x   Ord. High-PIXEL CURS.
21983  9b93
21984  9b93  68              pla             16, wenn negativ, sonst 18
21985  9b94  a8              tay
21986  9b95
21987  9b95  18              clc
21988  9b96  a9 08           lda #8
21989  9b98  6d 48 11        adc greatr      2, wenn negativ, sonst 0
21990  9b9b  aa              tax             10 , wenn negativ, sonst 8
21991  9b9c  20 7c 9d        jsr subtwo      Koord.(yr) minus Koord.(xr) nach
                                             ac/yr
21992  9b9f
21993  9b9f  8d 45 11        sta fct+4
21994  9ba2  8c 46 11        sty fct+5
21995  9ba5
21996  9ba5  20 fb 9b  dloop jsr gplot       Test/einfachen/dopp. Punkt setzen
21997  9ba8
21998  9ba8  ac 48 11        ldy greatr
21999  9bab  38              sec
22000  9bac  b9 39 11        lda xyabs,y
22001  9baf  e9 01           sbc #1
22002  9bb1  99 39 11        sta xyabs,y
22003  9bb4  b0 0b           bcs drw05
22004  9bb6
22005  9bb6  b9 3a 11        lda xyabs+1,y
22006  9bb9  e9 00           sbc #0
22007  9bbb  99 3a 11        sta xyabs+1,y
22008  9bbe  b0 01           bcs drw05
22009  9bc0
22010  9bc0  60              rts
22011  9bc1
22012  9bc1
22013  9bc1  ae 47 11  drw05 ldx lesser
22014  9bc4  ad 46 11        lda fct+5
```

```
22015  9bc7  30 06           bmi drw06
22016  9bc9
22017  9bc9  20 ea 9b        jsr drwinc      Increment addieren
22018  9bcc  ae 48 11        ldx greatr
22019  9bcf  18        drw06 clc
22020  9bd0  ad 45 11        lda fct+4
22021  9bd3  7d 41 11        adc fct,x
22022  9bd6  8d 45 11        sta fct+4
22023  9bd9
22024  9bd9  ad 46 11        lda fct+5
22025  9bdc  7d 42 11        adc fct+1,x
22026  9bdf  8d 46 11        sta fct+5
22027  9be2
22028  9be2  ae 48 11        ldx greatr
22029  9be5  20 ea 9b        jsr drwinc      Increment addieren
22030  9be8  f0 bb           beq dloop
22031  9bea
22032  9bea
22033  9bea  ===> Increment addieren <===
22034  9bea
22035  9bea  a0 02     drwinc ldy #2
22036  9bec  18               clc
22037  9bed  bd 31 11  drwin1 lda vwork,x
22038  9bf0  7d 3d 11         adc xysgn,x
22039  9bf3  9d 31 11         sta vwork,x
22040  9bf6
22041  9bf6  e8               inx
22042  9bf7  88               dey
22043  9bf8  d0 f3            bne drwin1
22044  9bfa
22045  9bfa  60               rts
22046  9bfb
22047  9bfb
22048  9bfb  ===> Test/einfachen/dopp. Punkt setzen <===
22049  9bfb
22050  9bfb  ad 6c 11  gplot  lda filflg      Flag 1=Ausmalen, 0=nicht
22051  9bfe  0d 6b 11         ora width       Strichstärke (0-1)
22052  9c01  f0 16           beq plot01      Graphik Punkt setzen
22053  9c03
22054  9c03  ee 31 11         inc vwork
22055  9c06  d0 03            bne gplo10
22056  9c08
22057  9c08  ee 32 11         inc xypos+1     Ord. High-PIXEL CURS.
22058  9c0b  20 19 9c  gplo10 jsr plot01      Graphik Punkt setzen
22059  9c0e  ae 31 11         ldx vwork
22060  9c11  d0 03            bne gplo20
22061  9c13
22062  9c13  ce 32 11         dec xypos+1     Ord. High-PIXEL CURS.
22063  9c16  ce 31 11  gplo20 dec vwork
22064  9c19
22065  9c19
22066  9c19  ===> Graphik Punkt setzen <===
22067  9c19
22068  9c19  20 24 9d  plot01 jsr divpos      BS-Zeile/BS-Spalte Graph.Curs. nach
                                              xr/yr
22069  9c1c  b0 24            bcs plotgo      nicht im Graphik Bereich
22070  9c1e
22071  9c1e  20 70 9c         jsr docolr      Farbquelle in Color RAM eintragen
```

```
22072  9c21  20 e8 9c          jsr getps1       Graph.pnt., Pix.Curs., Bitmaske
                                                berechnen
22073  9c24  8d 6d 11          sta bitmsk
22074  9c27
22075  9c27  b1 8c             lda (grapnt),y   ADDRL Zeiger in Graphik-RAM
22076  9c29  0d 6d 11          ora bitmsk
22077  9c2c  24 d8             bit graphm       Flag Text/Graphik/Split-BS
22078  9c2e  10 13             bpl plot30
22079  9c30
22080  9c30  ===> Multi Color <===
22081  9c30
22082  9c30  48                pha
22083  9c31  a6 83             ldx colsel       Farbquelle (0-3)
22084  9c33  ad 6d 11          lda bitmsk
22085  9c36  3d 25 9f          and colval,x     Tab. Bitmuster Farbquelle Sprites
22086  9c39  8d 6d 11          sta bitmsk
22087  9c3c  68                pla
22088  9c3d
22089  9c3d  4d 6d 11   plot10 eor bitmsk
22090  9c40  91 8c      plot20 sta (grapnt),y   ADDRL Zeiger in Graphik-RAM
22091  9c42
22092  9c42  60         plotgo rts
22093  9c43
22094  9c43
22095  9c43  a6 83      plot30 ldx colsel       Farbquelle (0-3)
22096  9c45  d0 f9             bne plot20
22097  9c47  f0 f4             beq plot10
22098  9c49
22099  9c49
22100  9c49  ===> Test auf Punkt xr/yr gesetzt <===
22101  9c49
22102  9c49  20 e3 9c   readpt jsr getpos       Graph.pnt., Pix.Curs., Bitmaske
                                                berechnen
22103  9c4c  b0 21             bcs readgo       nicht im Graphik Bereich
22104  9c4e
22105  9c4e  8d 6d 11          sta bitmsk
22106  9c51
22107  9c51  b1 8c             lda (grapnt),y   ADDRL Zeiger in Graphik-RAM
22108  9c53  2d 6d 11          and bitmsk
22109  9c56  2a         readlp rol a
22110  9c57  ca                dex
22111  9c58  10 fc             bpl readlp
22112  9c5a
22113  9c5a  2a                rol a
22114  9c5b  24 8b             bit stopnb       Flag für paint, Rahmenflag
22115  9c5d  30 06             bmi readl1
22116  9c5f
22117  9c5f  ===> Test gleiche Farbe <===
22118  9c5f
22119  9c5f  29 03             and #%00000011
22120  9c61  c5 83             cmp colsel       Farbquelle (0-3)
22121  9c63  18                clc
22122  9c64  60                rts
22123  9c65
22124  9c65
22125  9c65  18         readl1 clc
22126  9c66  29 03             and #%00000011
22127  9c68  f0 03             beq readl3       Punkt nicht gesetzt
```

```
22128  9c6a
22129  9c6a
22129  9c6a  ===> Punkt gesetzt <===
22130  9c6a
22131  9c6a  a2 00             ldx #$00
22132  9c6c  60                rts
22133  9c6d
22134  9c6d
22135  9c6d  ===> Punkt nicht gesetzt <===
22136  9c6d
22137  9c6d  a2 ff       readl3 ldx #$ff
22138  9c6f  60          readgo rts
22139  9c70
22140  9c70
22141  9c70  ===> Farbquelle in Color RAM eintragen <===
22142  9c70
22143  9c70  bd 33 c0    docolr lda ldtb2,x      Tab. ADDRL BS Zeilen
22144  9c73  85 8c              sta grapnt       ADDRL Zeiger in Graphik-RAM
22145  9c75
22146  9c75  bd ca 9c           lda gldtb1,x     Tab. ADDRH BS-Zeilen Color RAM
22147  9c78  85 8d              sta grapnt+1     ADDRH Zeiger in Graphik-RAM
22148  9c7a
22149  9c7a  a5 83              lda colsel       Farbquelle (0-3)
22150  9c7c  d0 08              bne docl10       Test Zusatzfarbe 1/2
22151  9c7e
22152  9c7e  ad e2 03           lda fgbg         VIC Vorder-/Hintergrund Farbe
22153  9c81  24 d8              bit graphm       Flag Text/Graphik/Split-BS
22154  9c83  10 08              bpl docl20
22155  9c85
22156  9c85  60                 rts
22157  9c86
22158  9c86
22159  9c86  ===> Test Zusatzfarbe 1/2 <===
22160  9c86
22161  9c86  c9 02       docl10 cmp #2
22162  9c88  d0 10              bne docl30       Test Zusatzfarbe 2/Vordergrund
22163  9c8a
22164  9c8a  ===> neue Hintergrund Farbe <===
22165  9c8a
22166  9c8a  ad e3 03           lda fgmc1        Zusatzfarbe 1
22167  9c8d  29 0f       docl20 and #%00001111   minus Vordergrund
22168  9c8f  85 77              sta mltus1       Hintergrund Farbe
22169  9c91
22170  9c91  b1 8c              lda (grapnt),y   Hintergrund/Vordergrund
22171  9c93  29 f0              and #%11110000   minus Hintergrund alt
22172  9c95  05 77              ora mltus1       plus Hintergrund neu
22173  9c97  91 8c              sta (grapnt),y   Hintergrund/Vordergrund neu
22174  9c99
22175  9c99  60                 rts
22176  9c9a
22177  9c9a
22178  9c9a  ===> Test Zusatzfarbe 2/Vordergrund <===
22179  9c9a
22180  9c9a  b0 10       docl30 bcs docl40       Zusatzfarbe 2 neu
22181  9c9c
22182  9c9c  ===> neue Vordergrund Farbe <===
22183  9c9c
22184  9c9c  ad e2 03           lda fgbg         VIC Vorder-/Hintergrund Farbe
```

```
22185  9c9f  29 f0           and #%11110000   minus Hintergrund
22186  9ca1  85 77           sta mltus1       Vordergrund Farbe
22187  9ca3
22188  9ca3  b1 8c           lda (grapnt),y   Hintergrund/Vordergrund
22189  9ca5  29 0f           and #%00001111   minus Vordergrund alt
22190  9ca7  05 77           ora mltus1       plus Vordergrund neu
22191  9ca9  91 8c           sta (grapnt),y   Hintergrund/Vordergrund neu
22192  9cab
22193  9cab  60              rts
22194  9cac
22195  9cac
22196  9cac  ===> Zusatzfarbe 2 neu <===
22197  9cac
22198  9cac  a5 8d     docl40 lda grapnt+1     ADDRH Zeiger in Graphik-RAM
22199  9cae  29 03            and #%00000011
22200  9cb0  09 d8            ora #>colrmh
22201  9cb2  85 8d            sta grapnt+1     ADDRH Zeiger in Graphik-RAM
22202  9cb4
22203  9cb4  a9 00            lda #%00000000   I/O Chips einblenden
22204  9cb6
22205  9cb6   00xxxxxx  RAM Bank 0
22206  9cb6   xx00xxxx  KERNAL ROM    $c000-$cfff
22207  9cb6   xxxx00xx  BASIC ROM hi  $8000-$bfff
22208  9cb6   xxxxxx0x  BASIC ROM lo  $4000-$7fff
22209  9cb6   xxxxxxx0  I/O-Chips     $d000-$dfff
22210  9cb6
22211  9cb6  8d 00 ff         sta mmucr        MMU 8722 Config. Reg. (Kopie)
22212  9cb9
22213  9cb9  78               sei
22214  9cba  a5 01            lda r8502        CPU 8502 Port Data Reg.
22215  9cbc  48               pha
22216  9cbd
22217  9cbd  29 fe            and #%11111110
22218  9cbf
22219  9cbf   xxxxxxx0  Color RAM hi ein $d800-$dbff
22220  9cbf
22221  9cbf  85 01            sta r8502        CPU 8502 Port Data Reg.
22222  9cc1
22223  9cc1  a5 85            lda mulco2       Zusatzfarbe 2
22224  9cc3  91 8c            sta (grapnt),y   ADDRL Zeiger in Graphik-RAM
22225  9cc5
22226  9cc5  68               pla
22227  9cc6  85 01            sta r8502        CPU 8502 Port Data Reg.
22228  9cc8
22229  9cc8  58               cli
22230  9cc9  60               rts
22231  9cca
22232  9cca
22233  9cca  ===> Tab. ADDRH BS-Zeilen Color RAM <===
22234  9cca
22235  9cca  1c          gldtb1 .byte >colrml    ADDRH BS-Zeile  1 Color RAM
22236  9ccb  1c                 .byte >colrml+40 ADDRH BS-Zeile  2 Color RAM
22237  9ccc  1c                 .byte >colrml+80 ADDRH BS-Zeile  3 Color RAM
22238  9ccd  1c                 .byte >colrml+120      ADDRH BS-Zeile  4 Color RAM
22239  9cce  1c                 .byte >colrml+160      ADDRH BS-Zeile  5 Color RAM
22240  9ccf  1c                 .byte >colrml+200      ADDRH BS-Zeile  6 Color RAM
22241  9cd0  1c                 .byte >colrml+240      ADDRH BS-Zeile  7 Color RAM
22242  9cd1  1d                 .byte >colrml+280      ADDRH BS-Zeile  8 Color RAM
```

```
22243  9cd2  1d                .byte >colrml+320      ADDRH BS-Zeile  9 Color RAM
22244  9cd3  1d                .byte >colrml+360      ADDRH BS-Zeile 10 Color RAM
22245  9cd4  1d                .byte >colrml+400      ADDRH BS-Zeile 11 Color RAM
22246  9cd5  1d                .byte >colrml+440      ADDRH BS-Zeile 12 Color RAM
22247  9cd6  1d                .byte >colrml+480      ADDRH BS-Zeile 13 Color RAM
22248  9cd7  1e                .byte >colrml+520      ADDRH BS-Zeile 14 Color RAM
22249  9cd8  1e                .byte >colrml+560      ADDRH BS-Zeile 15 Color RAM
22250  9cd9  1e                .byte >colrml+600      ADDRH BS-Zeile 16 Color RAM
22251  9cda  1e                .byte >colrml+640      ADDRH BS-Zeile 17 Color RAM
22252  9cdb  1e                .byte >colrml+680      ADDRH BS-Zeile 18 Color RAM
22253  9cdc  1e                .byte >colrml+720      ADDRH BS-Zeile 19 Color RAM
22254  9cdd  1e                .byte >colrml+760      ADDRH BS-Zeile 20 Color RAM
22255  9cde  1f                .byte >colrml+800      ADDRH BS-Zeile 21 Color RAM
22256  9cdf  1f                .byte >colrml+840      ADDRH BS-Zeile 22 Color RAM
22257  9ce0  1f                .byte >colrml+880      ADDRH BS-Zeile 23 Color RAM
22258  9ce1  1f                .byte >colrml+920      ADDRH BS-Zeile 24 Color RAM
22259  9ce2  1f                .byte >colrml+960      ADDRH BS-Zeile 25 Color RAM
22260  9ce3
22261  9ce3
22262  9ce3  ===> Graph.pnt., Pix.Curs., Bitmaske berechnen <===
22263  9ce3
22264  9ce3  20 24 9d   getpos jsr divpos       BS-Zeile/BS-Spalte Graph.Curs. nach
                                                xr/yr
22265  9ce6  b0 33             bcs grprts       nicht in Graphik-Bereich
22266  9ce8
22267  9ce8
22268  9ce8  ===> Graph.pnt., Pix.Curs., Bitmaske berechnen <===
22269  9ce8
22270  9ce8  98         getps1 tya              Spalte
22271  9ce9  18                clc
22272  9cea  7d 33 c0          adc ldtb2,x      Tab. ADDRL BS Zeilen
22273  9ced  85 8c             sta grapnt       Low-Bytezeiger
22274  9cef
22275  9cef  bd 4c c0          lda ldtb1,x      Tab. ADDRH BS Zeilen
22276  9cf2  69 00             adc #0           High-Bytezeiger
22277  9cf4
22278  9cf4  ===> Bytenummer mal acht = Bitzeiger <===
22279  9cf4
22280  9cf4  06 8c             asl grapnt       ADDRL Zeiger in Graphik-RAM
22281  9cf6  2a                rol a
22282  9cf7
22283  9cf7  06 8c             asl grapnt       ADDRL Zeiger in Graphik-RAM
22284  9cf9  2a                rol a
22285  9cfa
22286  9cfa  06 8c             asl grapnt       ADDRL Bit Zeiger in Graphik-RAM
22287  9cfc  2a                rol a
22288  9cfd  85 8d             sta grapnt+1     ADDRH Bit Zeiger in Graphik-RAM
22289  9cff
22290  9cff  ===> Offset Koordinaten berechnen <===
22291  9cff
22292  9cff  ad 33 11          lda xypos+2      Absz. Low-PIXEL CURS.
22293  9d02  29 07             and #%00000111
22294  9d04  a8                tay              Offset Ordinate
22295  9d05
22296  9d05  ad 31 11          lda vwork
22297  9d08
22298  9d08  24 d8             bit graphm       Flag Text/Graphik/Split-BS
22299  9d0a  08                php
```

```
22300  9d0b  10 01            bpl grpos3
22301  9d0d
22302  9d0d  0a               asl a           mal zwei
22303  9d0e  29 07      grpos3 and #%00000111
22304  9d10  aa               tax             Offset Abszisse
22305  9d11
22306  9d11  bd 1c 9d         lda rbits,x     Tab. Bits7-0
22307  9d14  28               plp
22308  9d15  10 04            bpl grprts
22309  9d17
22310  9d17  ===> Multi Color <===
22311  9d17
22312  9d17  e8               inx
22313  9d18  1d 1c 9d         ora rbits,x     Tab. Bits7-0
22314  9d1b
22315  9d1b  60         grprts rts
22316  9d1c
22317  9d1c
22318  9d1c  ===> Tab. Bits7-0 <===
22319  9d1c
22320  9d1c  80 40      rbits  .byte %10000000, %01000000
22321  9d1e  20 10             .byte %00100000, %00010000
22322  9d20  08 04             .byte %00001000, %00000100
22323  9d22  02 01             .byte %00000010, %00000001
22324  9d24
22325  9d24
22326  9d24                    .end
22327  9d24                    .lib basic.graph_5
22328  9d24  put"@basic.graph_5"
```

```
22330  9d24
22331  9d24
22332  9d24  ===> BS-Zeile/BS-Spalte Graph.Curs. nach xr/yr <===
22333  9d24
22334  9d24  ad 32 11   divpos lda xypos+1      Ord. High-PIXEL CURS.
22335  9d27  4a               lsr a           Test > 511
22336  9d28  d0 1e            bne divp20      nicht im Graphik Bereich
22337  9d2a
22338  9d2a  ===> Ordinate durch vier / acht <===
22339  9d2a
22340  9d2a  ad 31 11         lda vwork
22341  9d2d  6a               ror a
22342  9d2e  4a               lsr a
22343  9d2f
22344  9d2f  24 d8            bit graphm      Flag Text/Graphik/Split-BS
22345  9d31  30 01            bmi divp10
22346  9d33
22347  9d33  4a               lsr a
22348  9d34
22349  9d34  a8         divp10 tay             Spalte
22350  9d35  c0 28            cpy #40         höchste Spalte +1
22351  9d37  b0 0f            bcs divp20      nicht im Graphik Bereich
22352  9d39
22353  9d39  ad 34 11         lda xypos+3     Absz. High-PIXEL CURS.
22354  9d3c  d0 0a            bne divp20      viel zu hoch
22355  9d3e
22356  9d3e  ===> Abszisse durch acht <===
22357  9d3e
22358  9d3e  ad 33 11         lda xypos+2     Absz. Low-PIXEL CURS.
22359  9d41  4a               lsr a
22360  9d42  4a               lsr a
22361  9d43  4a               lsr a
22362  9d44  aa               tax
22363  9d45
22364  9d45  c9 19            cmp #25         höchste Zeile +1
22365  9d47  60               rts
22366  9d48
22367  9d48
22368  9d48  ===> nicht im Graphik Bereich <===
22369  9d48
22370  9d48  38         divp20 sec
22371  9d49  60               rts
22372  9d4a
22373  9d4a
22374  9d4a  ===> Test/SCALE: Koord. umrechnen <===
22375  9d4a
22376  9d4a  ad 6a 11   scalxy lda scalem       SCALE Flag
22377  9d4d  f0 17            beq sclrts      SCALE 0
22378  9d4f
22379  9d4f  ===> SCALE: Koord. umrechnen <===
22380  9d4f
22381  9d4f  a5 87            lda scalex      SCALE Low-Byte x-Wert (Abszisse)
22382  9d51  a4 88            ldy scalex+1    SCALE High-Byte x-Wert (Abszisse)
22383  9d53  20 5a 9d         jsr doscal      eine Koordinate umrechnen
22384  9d56
22385  9d56  a5 89            lda scaley      SCALE Low-Byte y-Wert (Ordinate)
22386  9d58  a4 8a            ldy scaley+1    SCALE High-Byte y-Wert (Ordinate)
22387  9d5a
```

```
22388  9d5a
22389  9d5a  ===> eine Koordinate umrechnen <===
22390  9d5a
22391  9d5a  20 ae 9d   doscal jsr twobyt       Koord.(xr) mal ac/yr durch 65536
22392  9d5d  9d 31 11          sta vwork,x
22393  9d60
22394  9d60  98                tya
22395  9d61  e8                inx
22396  9d62  9d 31 11          sta vwork,x
22397  9d65
22398  9d65  e8                inx
22399  9d66  60         sclrts rts
22400  9d67
22401  9d67
22402  9d67  ===> ac/yr plus/minus Koord.(xr) nach ac/yr <===
22403  9d67
22404  9d67  90 07      dotwo2 bcc addtw2       ac/yr plus Koord.(xr) nach ac/yr
22405  9d69  b0 14              bcs subtw2       ac/yr minus Koord.(xr) nach ac/yr
22406  9d6b
22407  9d6b
22408  9d6b  ===> Koord.(yr) plus/minus Koord.(xr) nach ac/yr <===
22409  9d6b
22410  9d6b  b0 0f      dotwo  bcs subtwo       Koord.(yr) minus Koord.(xr) nach
                                                ac/yr
22411  9d6d
22412  9d6d
22413  9d6d  ===> Koord.(yr) plus Koord.(xr) nach ac/yr <===
22414  9d6d
22415  9d6d  20 8f 9d   addtwo jsr settwo       Winkel (yr) nach ac/yr
22416  9d70
22417  9d70
22418  9d70  ===> ac/yr plus Koord.(xr) nach ac/yr <===
22419  9d70
22420  9d70  18         addtw2 clc
22421  9d71  7d 31 11          adc vwork,x
22422  9d74  48                pha
22423  9d75
22424  9d75  98                tya
22425  9d76  7d 32 11          adc xypos+1,x    Ord. High-PIXEL CURS.
22426  9d79  a8                tay
22427  9d7a  68                pla
22428  9d7b  60                rts
22429  9d7c
22430  9d7c
22431  9d7c  ===> Koord.(yr) minus Koord.(xr) nach ac/yr <===
22432  9d7c
22433  9d7c  20 8f 9d   subtwo jsr settwo       Winkel (yr) nach ac/yr
22434  9d7f
22435  9d7f
22436  9d7f  ===> ac/yr minus Koord.(xr) nach ac/yr <===
22437  9d7f
22438  9d7f  38         subtw2 sec
22439  9d80  fd 31 11          sbc vwork,x
22440  9d83  85 59             sta tempf1       TMP FAC1
22441  9d85
22442  9d85  98                tya
22443  9d86  fd 32 11          sbc xypos+1,x    Ord. High-PIXEL CURS.
22444  9d89  a8                tay
```

```
22445  9d8a  08              php
22446  9d8b  a5 59           lda tempf1      TMP FAC1
22447  9d8d  28              plp
22448  9d8e  60              rts
22449  9d8f
22450  9d8f
22451  9d8f  ===> Winkel (yr) nach ac/yr <===
22452  9d8f
22453  9d8f  b9 31 11  settwo lda vwork,y
22454  9d92  48              pha
22455  9d93
22456  9d93  b9 32 11        lda xypos+1,y   Ord. High-PIXEL CURS.
22457  9d96  a8              tay
22458  9d97  68              pla
22459  9d98  60              rts
22460  9d99
22461  9d99
22462  9d99  ===> Abs.wert Koord.Diff. (yr,xr) nach ac, yr <===
22463  9d99
22464  9d99  20 7c 9d  abstwo jsr subtwo      Koord.(yr) minus Koord.(xr) nach
                                              ac/yr
22465  9d9c
22466  9d9c
22467  9d9c  ===> ac/yr invertieren, wenn neg. <===
22468  9d9c
22469  9d9c  10 0f     abstw2 bpl absrts
22470  9d9e
22471  9d9e
22472  9d9e  ===> neg. Differenz invertieren <===
22473  9d9e
22474  9d9e  08        invert php            PLUS/MINUS
22475  9d9f
22476  9d9f  18              clc
22477  9da0  49 ff           eor #$ff
22478  9da2  69 01           adc #1
22479  9da4  48              pha
22480  9da5
22481  9da5  98              tya
22482  9da6  49 ff           eor #$ff
22483  9da8  69 00           adc #0
22484  9daa
22485  9daa  a8              tay             High-Byte Abs.Wert
22486  9dab  68              pla             Low-Byte Abs.Wert
22487  9dac
22488  9dac  28              plp             PLUS/MINUS
22489  9dad
22490  9dad  60        absrts rts
22491  9dae
22492  9dae
22493  9dae  ===> Koord.(xr) mal ac/yr durch 65536 <===
22494  9dae
22495  9dae  84 8e     twobyt sty vtemp1      High-Byte Faktor 1
22496  9db0  85 8f            sta vtemp2      Low-Byte Faktor 1
22497  9db2
22498  9db2  bd 31 11        lda vwork,x     Low-Byte Faktor 2
22499  9db5  bc 32 11        ldy xypos+1,x   High-Byte Faktor 2
22500  9db8  08              php
22501  9db9
```

```
22502  9db9  20 9c 9d          jsr abstw2       Absolutwert Faktor 2
22503  9dbc  9d 31 11          sta vwork,x      ABS Low-Byte Faktor 2
22504  9dbf  98                tya
22505  9dc0  9d 32 11          sta xypos+1,x    ABS High-Byte Faktor 2
22506  9dc3
22507  9dc3  a9 00             lda #0           Low-Byte Produkt
22508  9dc5  8d 77 11          sta vtemp3       High-Byte Produkt
22509  9dc8
22510  9dc8  ===> 16 Bit Multiplikation <===
22511  9dc8
22512  9dc8  a0 10             ldy #16          Bit Zähler
22513  9dca  46 8e      twoby1 lsr vtemp1
22514  9dcc  66 8f             ror vtemp2
22515  9dce  90 0f             bcc twoby2       akt. Bit =0
22516  9dd0
22517  9dd0  ===> akt. Bit =1 <===
22518  9dd0
22519  9dd0  18                clc
22520  9dd1  7d 31 11          adc vwork,x
22521  9dd4  48                pha              Low-Byte Zwischenergebnis
22522  9dd5
22523  9dd5  ad 77 11          lda vtemp3
22524  9dd8  7d 32 11          adc xypos+1,x    Ord. High-PIXEL CURS.
22525  9ddb  8d 77 11          sta vtemp3       High-Byte Zwischenergebnis
22526  9dde  68                pla              Low-Byte Zwischenergebnis
22527  9ddf
22528  9ddf  4e 77 11   twoby2 lsr vtemp3       High-Byte Zwischenergebnis
22529  9de2  6a                ror a            Low-Byte Zwischenergebnis
22530  9de3
22531  9de3  88                dey
22532  9de4  d0 e4             bne twoby1
22533  9de6
22534  9de6  69 00             adc #0           Low-Byte Produkt
22535  9de8  ac 77 11          ldy vtemp3       High-Byte Produkt
22536  9deb  90 01             bcc tworts
22537  9ded  c8                iny
22538  9dee
22539  9dee  28         tworts plp
22540  9def  4c 9c 9d          jmp abstw2       ac/yr invertieren, wenn neg.
22541  9df2
22542  9df2
22543  9df2  ===> Zielkoord. werden akt. Koord. <===
22544  9df2
22545  9df2  a0 00      dstpos ldy #xdest-xdest
22546  9df4  20 f9 9d          jsr dstmov       eine Zielkoord. nach akt. Koord.
22547  9df7
22548  9df7  a0 02             ldy #ydest-xdest
22549  9df9
22550  9df9
22551  9df9  ===> eine Zielkoord. nach akt. Koord. <===
22552  9df9
22553  9df9  b9 35 11   dstmov lda xdest,y
22554  9dfc  99 31 11          sta vwork,y
22555  9dff
22556  9dff  b9 36 11          lda xdest+1,y
22557  9e02  99 32 11          sta xypos+1,y    Ord. High-PIXEL CURS.
22558  9e05
22559  9e05  60                rts
```

```
22560  9e06
22561  9e06
22562  9e06
22562  9e06  ===> opt. 16 Bit INT nach yr/ac, sonst 0 <===
22563  9e06
22564  9e06  20 86 03   optwrd jsr chrgot       letztes Zeichen erneut nach ac
22565  9e09  f0 0c             beq optw10       Ersatzwert 0 nach yr/ac
22566  9e0b
22567  9e0b  20 5c 79         jsr chkcom       Test auf folg. »Komma« sonst Fehler
22568  9e0e  c9 2c            cmp #','
22569  9e10  f0 05            beq optw10       Ersatzwert 0 nach yr/ac
22570  9e12
22571  9e12  20 12 88         jsr getwrd       16 Bit INT nach linnum
22572  9e15  38               sec              Flag Parameter gelesen
22573  9e16  60               rts
22574  9e17
22575  9e17
22576  9e17  ===> Ersatzwert 0 nach yr/ac <===
22577  9e17
22578  9e17  a9 00      optw10 lda #0           High-Byte
22579  9e19  a8               tay              Low-Byte
22580  9e1a
22581  9e1a  18         optw99 clc             Flag Ersatzwert
22582  9e1b  60               rts
22583  9e1c
22584  9e1c
22585  9e1c  ===> opt. 8 Bit INT nach xr, sonst 0 <===
22586  9e1c
22587  9e1c  a2 00      optzer ldx #0          Vorbelegung xr mit 0
22588  9e1e
22589  9e1e
22590  9e1e  ===> opt. 8 Bit INT nach xr <===
22591  9e1e
22592  9e1e  20 86 03   optbyt jsr chrgot       letztes Zeichen erneut nach ac
22593  9e21  f0 f7            beq optw99       Trennzeichen
22594  9e23
22595  9e23  20 5c 79         jsr chkcom       Test auf folg. »Komma« sonst Fehler
22596  9e26  c9 2c            cmp #','
22597  9e28  f0 f0            beq optw99
22598  9e2a
22599  9e2a  20 f4 87         jsr getbyt       8 Bit INT üb. frmnum nach xr
22600  9e2d
22601  9e2d  38               sec              Flag Parameter gelesen
22602  9e2e  60               rts
22603  9e2f
22604  9e2f
22605  9e2f  ===> Test GRAPHIC, Farbqu. nach colsel <===
22606  9e2f
22607  9e2f  20 74 a0   grpcol jsr isgrap       Test GRAPHIC-RAM reserviert
22608  9e32
22609  9e32
22610  9e32  ===> Farbquelle (0-3) einlesen <===
22611  9e32
22612  9e32  a2 01      incolr ldx #1          Vorgabe Vordergrund (Zeichen)
22613  9e34
22614  9e34  20 86 03         jsr chrgot       letztes Zeichen erneut nach ac
22615  9e37  f0 13            beq incol2       Trennzeichen, Farbqu. = Vorgabe
22616  9e39
```

```
22617  9e39  c9 2c            cmp #','
22618  9e3b  f0 0f            beq incol2      kein Param., Farbqu. = Vorgabe
22619  9e3d
22620  9e3d
22620  9e3d  ===> Farbquelle einlesen <===0
22621  9e3d
22622  9e3d  20 f4 87          jsr getbyt      Farbquelle (0-3) nach xr
22623  9e40  e0 04            cpx #4
22624  9e42  b0 0b            bcs illval      Behandlung »?illegal quantity error«
22625  9e44
22626  9e44  e0 02            cpx #2          Zusatzfarbe 2-3
22627  9e46
22628  9e46  24 d8            bit graphm      Graphikmodus Bit7=1
22629  9e48  30 02            bmi incol2      Farbquelle abspeichern
22630  9e4a
22631  9e4a  b0 03            bcs illval      Behandlung »?illegal quantity error«
22632  9e4c
22633  9e4c
22634  9e4c  ===> Farbquelle abspeichern <===
22635  9e4c
22636  9e4c  86 83      incol2 stx colsel      Farbquelle (0-3)
22637  9e4e
22638  9e4e  60               rts
22639  9e4f
22640  9e4f
22641  9e4f  ===> Behandlung »?illegal quantity error« <===
22642  9e4f
22643  9e4f  4c 28 7d   illval jmp fcerr       Behandlung »?illegal quantity error«
22644  9e52
22645  9e52
22646  9e52  ===> Koordinaten einlesen <===
22647  9e52
22648  9e52   Koordinaten können auf versch. Weise angegeben werden:
22649  9e52
22650  9e52   1. keine Angabe (aber Komma):        akt. PIXEL Cursor Wert
22651  9e52   2. Koord. durch Komma getrennt:      skalierte Koord.
22652  9e52   3. + oder - vor Koordinaten:         skal. rel. zu PIXEL Cursor
22653  9e52   4. Koord. durch Semikolon getrennt:  polar rel. zu PIXEL Cursor
22654  9e52
22655  9e52
22656  9e52  ===> Ersatz-/Start-/Zielkoord. einlesen <===
22657  9e52
22658  9e52  20 86 03   in2cor jsr chrgot      letztes Zeichen erneut nach ac
22659  9e55  f0 07            beq defcor      Vorgabe Koord. PIXEL CURS.
22660  9e57
22661  9e57  20 5c 79          jsr chkcom      Test auf folg. »Komma« sonst Fehler
22662  9e5a  c9 2c            cmp #','
22663  9e5c  d0 12            bne incord      Start-/Zielkoord. einlesen
22664  9e5e
22665  9e5e
22666  9e5e  ===> Vorgabe Koord. PIXEL CURS. <===
22667  9e5e
22668  9e5e  a0 00      defcor ldy #0         Zeiger auf Koord. PIXEL CURS.
22669  9e60  b9 31 11   incr2c lda vwork,y
22670  9e63  9d 31 11          sta vwork,x
22671  9e66
22672  9e66  e8               inx
22673  9e67  c8               iny
```

```
22674  9e68  c0 04            cpy #4
22675  9e6a  d0 f4            bne incr2c
22676  9e6c
22677  9e6c  60               rts
22678  9e6d
22679  9e6d
22680  9e6d  ===> Kommatest, Start-/Zielkoord. einl. <===
22681  9e6d
22682  9e6d  20 5c 79   incor3 jsr chkcom       Test auf folg. »Komma« sonst Fehler
22683  9e70
22684  9e70
22685  9e70  ===> Start-/Zielkoord. einlesen <===
22686  9e70
22687  9e70  8e 78 11   incord stx vtemp4       Zeiger in vwork
22688  9e73  20 08 9f          jsr cordsb       (rel.) Koord. nach vwork+vtemp4
22689  9e76
22690  9e76  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
22691  9e79  c9 2c            cmp #','
22692  9e7b  f0 56            beq docord       (rel.) rechtw. Koordinaten berechnen
22693  9e7d
22694  9e7d  c9 3b            cmp #';'
22695  9e7f  f0 03            beq polcrd       Polar Koordinaten berechnen
22696  9e81
22697  9e81  ===> Bearbeitung »?syntax error« <===
22698  9e81
22699  9e81  4c 6c 79          jmp snerr        Behandlung »?syntax error«
22700  9e84
22701  9e84
22702  9e84  ===> Polar Koordinaten berechnen <===
22703  9e84
22704  9e84  20 80 03   polcrd jsr chrget       nächstes Zeichen nach ac
22705  9e87  20 12 88          jsr getwrd       Winkel (dez.) nach yr/ac
22706  9e8a
22707  9e8a  85 77            sta mltus1       yr/ac (low/high) tauschen
22708  9e8c  98               tya
22709  9e8d  a4 77            ldy mltus1
22710  9e8f  20 77 9a          jsr gtang1       SIN/COS ac/yr nach sinpc/cospc
22711  9e92
22712  9e92  ae 78 11          ldx vtemp4       Zeiger in vwork
22713  9e95  bd 31 11          lda vwork,x
22714  9e98  9d 33 11          sta xypos+2,x    Absz. Low-PIXEL CURS.
22715  9e9b
22716  9e9b  bd 32 11          lda xypos+1,x    Ord. High-PIXEL CURS.
22717  9e9e  9d 34 11          sta xypos+3,x    Absz. High-PIXEL CURS.
22718  9ea1
22719  9ea1  20 4a 9d          jsr scalxy       Test/SCALE: Koord. umrechnen
22720  9ea4
22721  9ea4  a9 0e            lda #%00001110   SIN/COS Flags
22722  9ea6  8d 79 11          sta vtemp5
22723  9ea9
22724  9ea9  18               clc
22725  9eaa  ae 78 11          ldx vtemp4       Zeiger in vwork
22726  9ead  20 ce 9a   doang1 jsr angmlt       Koord.(xr) mal SIN/COS nach ac/yr
22727  9eb0  9d 31 11          sta vwork,x
22728  9eb3
22729  9eb3  98               tya
22730  9eb4  9d 32 11          sta xypos+1,x    Ord. High-PIXEL CURS.
22731  9eb7
```

```
22732  9eb7  a0 00           ldy #0
22733  9eb9  4e 79 11        lsr vtemp5
22734  9ebc  90 02           bcc doang2
22735  9ebe
22736  9ebe  a0 02           ldy #2
22737  9ec0  20 6b 9d  doang2 jsr dotwo       Koord.(yr) plus/minus Koord.(xr)
                                              nach ac/yr
22738  9ec3  9d 31 11        sta vwork,x
22739  9ec6  98              tya
22740  9ec7  9d 32 11        sta xypos+1,x    Ord. High-PIXEL CURS.
22741  9eca
22742  9eca  e8              inx
22743  9ecb  e8              inx
22744  9ecc  4e 79 11        lsr vtemp5
22745  9ecf  d0 dc           bne doang1
22746  9ed1
22747  9ed1  18              clc
22748  9ed2  60              rts
22749  9ed3
22750  9ed3
22751  9ed3  ===> (rel.) rechtw. Koordinaten berechnen <===
22752  9ed3
22753  9ed3  20 80 03  docord jsr chrget      nächstes Zeichen nach ac
22754  9ed6  ee 78 11        inc vtemp4       Zeiger in vwork
22755  9ed9  ee 78 11        inc vtemp4       Zeiger in vwork
22756  9edc  20 08 9f        jsr cordsb       (rel.) Koord. nach vwork+vtemp4
22757  9edf  ae 78 11        ldx vtemp4       Zeiger in vwork
22758  9ee2  ca              dex
22759  9ee3  ca              dex
22760  9ee4  20 4a 9d        jsr scalxy       Test/SCALE: Koord. umrechnen
22761  9ee7
22762  9ee7  a0 02           ldy #2
22763  9ee9  ae 78 11        ldx vtemp4       Zeiger in vwork
22764  9eec  e8              inx
22765  9eed  e8              inx
22766  9eee  ca        docor1 dex
22767  9eef  ca              dex
22768  9ef0  4e 79 11        lsr vtemp5
22769  9ef3  90 0a           bcc docor2
22770  9ef5
22771  9ef5  ===> rel. rechtw. Koordinaten <===
22772  9ef5
22773  9ef5  20 6d 9d        jsr addtwo       Koord.(yr) plus Koord.(xr) nach
                                              ac/yr
22774  9ef8  9d 31 11        sta vwork,x
22775  9efb
22776  9efb  98              tya
22777  9efc  9d 32 11        sta xypos+1,x    Ord. High-PIXEL CURS.
22778  9eff
22779  9eff  a0 00     docor2 ldy #0
22780  9f01  ec 78 11        cpx vtemp4       Zeiger in vwork
22781  9f04  f0 e8           beq docor1
22782  9f06
22783  9f06  18              clc
22784  9f07  60              rts
22785  9f08
22786  9f08
```

```
22787  9f08  ===> (rel.) Koord. nach vwork+vtemp4 <===
22788  9f08
22789  9f08  20 86 03   cordsb jsr chrgot       letztes Zeichen erneut nach ac
22790  9f0b  c9 aa             cmp #_plust      PLUS Token
22791  9f0d  f0 05             beq crdsb1       Flag setzen rel./abs. Koord.
22792  9f0f
22793  9f0f  c9 ab             cmp #_minut      MINUS Token
22794  9f11  f0 01             beq crdsb1       Flag setzen rel./abs. Koord.
22795  9f13
22796  9f13  18                clc              Flag für Absolut Koord.
22797  9f14
22798  9f14
22799  9f14  ===> Flag setzen rel./abs. Koord. <===
22800  9f14
22801  9f14  2e 79 11   crdsb1 rol vtemp5       Carry ins Flag
22802  9f17
22803  9f17  20 12 88          jsr getwrd       16 Bit INT nach linnum
22804  9f1a
22805  9f1a  ae 78 11          ldx vtemp4       Zeiger in vwork
22806  9f1d  9d 32 11          sta xypos+1,x    Ord. High-PIXEL CURS.
22807  9f20
22808  9f20  98                tya
22809  9f21  9d 31 11          sta vwork,x
22810  9f24
22811  9f24  60                rts
22812  9f25
22813  9f25
22814  9f25  ===> Tab. Bitmuster Farbquelle Sprites <===
22815  9f25
22816  9f25  ff         colval .byte %11111111  Hintergrund Farbe
22817  9f26  aa                .byte %10101010  Vordergrund Farbe
22818  9f27  55                .byte %01010101  Zusatzfarbe 1
22819  9f28  00                .byte %00000000  Zusatzfarbe 2
22820  9f29
22821  9f29
22822  9f29  ===> Tab. SIN Werte 0-90 Grad <===
22823  9f29
22824  9f29  00 00      angval .dbyte 0         65535 * 0.0000 (sin  0)
22825  9f2b  2c 71             .dbyte 11377     65535 * 0.1736 (sin 10)
22826  9f2d  57 8d             .dbyte 22413     65535 * 0.3420 (sin 20)
22827  9f2f  80 00             .dbyte 32768     65535 * 0.5000 (sin 30)
22828  9f31  a4 8f             .dbyte 42127     65535 * 0.6428 (sin 40)
22829  9f33  c4 19             .dbyte 50201     65535 * 0.7660 (sin 50)
22830  9f35  dd b2             .dbyte 56754     65535 * 0.8660 (sin 60)
22831  9f37  f0 90             .dbyte 61584     65535 * 0.9397 (sin 70)
22832  9f39  fc 1c             .dbyte 64540     65535 * 0.9848 (sin 80)
22833  9f3b  ff ff             .dbyte 65535     65535 * 1.0000 (sin 90)
22834  9f3d
22835  9f3d
22836  9f3d  ===> Tab. Interpolations Werte <===
22837  9f3d
22838  9f3d  04 72      incval .dbyte 1138
22839  9f3f  04 50             .dbyte 1104
22840  9f41  04 0b             .dbyte 1035
22841  9f43  03 a8             .dbyte 936
22842  9f45  03 28             .dbyte 808
22843  9f47  02 90             .dbyte 656
22844  9f49  01 e3             .dbyte 483
```

```
22845  9f4b  01 28            .dbyte 296
22846  9f4d  00 63            .dbyte 99
22847  9f4f
22848  9f4f
22849  9f4f  ===> Test / GRAPHIC RAM einrichten <===
22850  9f4f
22851  9f4f  a5 76      sethir lda mvdflg       Flag GRAPHIC-RAM
22852  9f51
22853  9f51   $00  kein GRAPHIC RAM eingerichtet
22854  9f51   $ff  GRAPHIC RAM eingerichtet
22855  9f51
22856  9f51  f0 01            beq sth010        Test 9KB frei / GRAPHIC RAM einr.
22857  9f53  60               rts
22858  9f54
22859  9f54
22860  9f54  ===> Test 9KB frei / GRAPHIC RAM einr. <===
22861  9f54
22862  9f54  ad 11 12   sth010 lda txttop+1     ADDRH Obergrenze akt. PGM
22863  9f57  18               clc
22864  9f58  69 24            adc #>_graph      9 KB RAM
22865  9f5a  b0 0e            bcs omerr7        Behandlung »?out of memory error«
22866  9f5c  85 62            sta lowtr+1
22867  9f5e
22868  9f5e  cd 13 12         cmp mxmem0+1      ADDRH Obergr. PGM RAM
22869  9f61  90 0a            bcc sth020        GRAPHIC RAM einrichten
22870  9f63  d0 05            bne omerr7        Behandlung »?out of memory error«
22871  9f65
22872  9f65  cc 12 12         cpy mxmem0        ADDRL Obergr. PGM RAM
22873  9f68  90 03            bcc sth020        GRAPHIC RAM einrichten
22874  9f6a
22875  9f6a  ===> Behandlung »?out of memeory error« <===0
22876  9f6a
22877  9f6a  4c 3a 4d   omerr7 jmp omerr        Behandlung »?out of memory error«
22878  9f6d
22879  9f6d
22880  9f6d  ===> GRAPHIC RAM einrichten <===
22881  9f6d
22882  9f6d  c6 76      sth020 dec mvdflg       Flag GRAPHIC-RAM
22883  9f6f
22884  9f6f   $ff  GRAPHIC RAM eingerichtet
22885  9f6f
22886  9f6f  ===> Zeiger auf neues PGM Ende <===
22887  9f6f
22888  9f6f  ad 10 12         lda txttop        ADDRL Obergrenze akt. PGM
22889  9f72  85 24            sta index1        ADDRL neue Obergrenze akt. PGM
22890  9f74
22891  9f74  a5 62            lda lowtr+1       ADDRH neue Obergrenze akt. PGM
22892  9f76  85 25            sta index1+1      ADDRH neue Obergrenze akt. PGM
22893  9f78
22894  9f78  ===> Zeiger auf altes PGM Ende <===
22895  9f78
22896  9f78  ae 10 12         ldx txttop        ADDRL Obergrenze akt. PGM
22897  9f7b  86 26            stx index2        ADDRL alte Obergrenze akt. PGM
22898  9f7d
22899  9f7d  ad 11 12         lda txttop+1      ADDRH Obergrenze akt. PGM
22900  9f80  85 27            sta index2+1      ADDRH alte Obergrenze akt. PGM
22901  9f82
```

```
22902  9f82  ===> (neg.) Programmlänge berechnen <===
22903  9f82
22904  9f82  38              sec
22905  9f83  e9 1c           sbc #>rambot     Untergrenze PGM RAM
22906  9f85  a8              tay              High-Byte PGM Länge
22907  9f86
22908  9f86  8a              txa
22909  9f87  49 ff           eor #$ff
22910  9f89  85 50           sta defpnt       Low-Byte neg. PGM Länge
22911  9f8b
22912  9f8b  98              tya
22913  9f8c  49 ff           eor #$ff
22914  9f8e  85 51           sta defpnt+1     High-Byte neg. PGM Länge
22915  9f90
22916  9f90  ===> akt. PGM um 9KB verschieben <===
22917  9f90
22918  9f90  a0 00           ldy #0
22919  9f92
22920  9f92
22921  9f92  ===> Test Verschiebung fertig <===
22922  9f92
22923  9f92  e6 50     sth030 inc defpnt      Low-Byte neg. PGM Länge
22924  9f94  d0 04            bne sth040      Speicherzeiger nachführen
22925  9f96
22926  9f96  e6 51            inc defpnt+1    High-Byte neg. PGM Länge
22927  9f98  f0 18            beq sth070      Verschiebung fertig
22928  9f9a
22929  9f9a
22930  9f9a  ===> Speicherzeiger nachführen <===
22931  9f9a
22932  9f9a  a5 24     sth040 lda index1      ADDRL neue Obergrenze akt. PGM
22933  9f9c  d0 02            bne sth050
22934  9f9e  c6 25            dec index1+1    ADDRH neue Obergrenze akt. PGM
22935  9fa0  c6 24     sth050 dec index1      ADDRL neue Obergrenze akt. PGM
22936  9fa2
22937  9fa2  ===> Ladezeiger nachführen <===
22938  9fa2
22939  9fa2  a5 26            lda index2      ADDRL alte Obergrenze akt. PGM
22940  9fa4  d0 02            bne sth060
22941  9fa6  c6 27            dec index2+1    ADDRH alte Obergrenze akt. PGM
22942  9fa8  c6 26     sth060 dec index2      ADDRL alte Obergrenze akt. PGM
22943  9faa
22944  9faa  20 c0 03         jsr ind2_0      PGM Byte nach ac
22945  9fad  91 24            sta (index1),y  PGM Byte abspeichern
22946  9faf
22947  9faf  4c 92 9f         jmp sth030      Test Verschiebung fertig
22948  9fb2
22949  9fb2
22950  9fb2  18        sth070 clc             kein Übertrag möglich
22951  9fb3
22952  9fb3  ad 11 12         lda txttop+1    ADDRH Obergrenze akt. PGM
22953  9fb6  69 24            adc #>_graph    + 9 KB
22954  9fb8  8d 11 12         sta txttop+1    ADDRH Obergrenze akt. PGM
22955  9fbb
22956  9fbb  a5 2e            lda txttab+1    ADDRH Untergrenze akt. PGM
22957  9fbd  69 24            adc #>_graph    + 9 KB
22958  9fbf  85 2e            sta txttab+1    ADDRH Untergrenze akt. PGM
22959  9fc1
```

```
22960  9fc1  a5 44           lda datptr+1     ADDRH Zeiger auf DATA (READ)
22961  9fc3  69 24           adc #>_graph     + 9 KB
22962  9fc5  85 44           sta datptr+1     ADDRH Zeiger auf DATA (READ)
22963  9fc7
22964  9fc7
22965  9fc7  ===> BASIC Zeiger um 9KB erhöhen <===
22966  9fc7
22967  9fc7  20 4f 4f  sth080 jsr lnkprg      alle Linkadressen neu berechnen
22968  9fca  20 82 4f        jsr fixlnk       Obergrenze akt. PGM nachführen
22969  9fcd
22970  9fcd  24 7f           bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
22971  9fcf  10 2d           bpl sth110
22972  9fd1
22973  9fd1  a2 24           ldx #>_graph     + 9 KB
22974  9fd3  24 76           bit mvdflg       Flag GRAPHIC-RAM
22975  9fd5  30 02           bmi sth090       BASIC Zeiger +/- 9 KB
22976  9fd7
22977  9fd7  a2 dc           ldx #>$ffff-_graph+1   - 9 KB
22978  9fd9
22979  9fd9
22980  9fd9  ===> BASIC Zeiger +/- 9 KB <===
22981  9fd9
22982  9fd9  8a        sth090 txa             +/- 9 KB
22983  9fda  18               clc
22984  9fdb  65 3e            adc txtptr+1     ADDRH chrget-Zeiger
22985  9fdd  85 3e            sta txtptr+1     ADDRH chrget-Zeiger
22986  9fdf
22987  9fdf  8a               txa              +/- 9 KB
22988  9fe0  18               clc
22989  9fe1  6d 03 12         adc oldtxt+1     ADDRH Zeiger letzter BASIC-Befehl
22990  9fe4  8d 03 12         sta oldtxt+1     ADDRH Zeiger letzter BASIC-Befehl
22991  9fe7
22992  9fe7  8a               txa              +/- 9 KB
22993  9fe8  18               clc
22994  9fe9  6d 0f 12         adc errtxt+1     ADDRH txtptr bei Fehler
22995  9fec  8d 0f 12         sta errtxt+1     ADDRH txtptr bei Fehler
22996  9fef
22997  9fef  20 47 50         jsr movtos       Kopie Stapelzeiger nach Suchzeiger
22998  9ff2
22999  9ff2
23000  9ff2  ===> Test / Korrektur BASIC Stapel <===
23001  9ff2
23002  9ff2  a5 3f     sth100 lda fndpnt       ADDRL BASIC-Stapel Suchzeiger
23003  9ff4  c9 ff            cmp #<rstvec-1   Restart-Vector
23004  9ff6  d0 07            bne sth120       Korrektur BASIC Stapel
23005  9ff8
23006  9ff8  a5 40            lda fndpnt+1     ADDRH BASIC-Stapel Suchzeiger
23007  9ffa  c9 09            cmp #>rstvec-1   Restart-Vector
23008  9ffc  d0 01            bne sth120       BASIC Stapel nicht leer
23009  9ffe
23010  9ffe  60        sth110 rts
23011  9fff
23012  9fff
23013  9fff  ===> Korrektur BASIC Stapel <===
23014  9fff
23015  9fff  a0 00     sth120 ldy #0           Zeiger auf Token in Stapel
23016  a001  b1 3f            lda (fndpnt),y   ADDRL BASIC-Stapel Suchzeiger
23017  a003
```

```
23018  a003  c9 81            cmp #_fort       FOR Token
23019  a005  d0 09            bne sth130
23020  a007
23021  a007  ===> Startadresse FOR Schleife auf Stapel korr. <===
23022  a007
23023  a007  a0 10            ldy #16          Zeiger auf ADDRH Startadr. Schleife
23024  a009  20 62 a0         jsr addoff       ADDRH aus BASIC Stapel korr.
23025  a00c
23026  a00c  a9 12            lda #_lnfor      Anzahl Stapelbytes FOR-NEXT
23027  a00e  d0 07            bne sth140       immer
23028  a010
23029  a010
23030  a010  a0 04      sth130 ldy #4           Zeiger auf ADDRH Rücksprungadresse
23031  a012  20 62 a0         jsr addoff       ADDRH aus BASIC Stapel korr.
23032  a015
23033  a015  a9 05            lda #_lngos      Anzahl Stapelbytes GOSUB
23034  a017
23035  a017  18        sth140 clc
23036  a018  65 3f            adc fndpnt       ADDRL BASIC-Stapel Suchzeiger
23037  a01a  85 3f            sta fndpnt       ADDRL BASIC-Stapel Suchzeiger
23038  a01c  90 d4            bcc sth100       Test / Korrektur BASIC Stapel
23039  a01e  e6 40            inc fndpnt+1     ADDRH BASIC-Stapel Suchzeiger
23040  a020  d0 d0            bne sth100       immer
23041  a022
23042  a022
23043  a022  ===> Test / GRAPHIC RAM freigeben <===
23044  a022
23045  a022  a5 76      clrhir lda mvdflg      Flag GRAPHIC-RAM
23046  a024
23047  a024   $00  kein GRAPHIC RAM eingerichtet
23048  a024   $ff  GRAPHIC RAM eingerichtet
23049  a024
23050  a024  d0 01            bne clh010       GRAPHIC RAM freigeben
23051  a026  60              rts
23052  a027
23053  a027
23054  a027  ===> GRAPHIC RAM freigeben <===
23055  a027
23056  a027  a0 00      clh010 ldy #$00         Flag GRAPHIC RAM nicht eingerichtet
23057  a029  84 76            sty mvdflg       Flag GRAPHIC-RAM
23058  a02b
23059  a02b  ===> Verschiebezeiger einrichten <===
23060  a02b
23061  a02b  84 24            sty index1       ADDRL neue Untergrenze
23062  a02d  84 26            sty index2       ADDRL alte Untergrenze
23063  a02f
23064  a02f  a9 1c            lda #>rambot     Untergr. PGM RAM ohne Graphik
23065  a031  85 25            sta index1+1     ADDRH neue Untergrenze
23066  a033
23067  a033  a9 40            lda #>gbasic     Untergr. PGM RAM mit Graphik
23068  a035  85 27            sta index2+1     ADDRH alte Untergrenze
23069  a037
23070  a037
23071  a037  ===> akt. PGM verschieben <===
23072  a037
23073  a037  20 c0 03   clh020 jsr ind2_0       PGM Byte nach ac
23074  a03a  91 24            sta (index1),y   PGM Byte abspeichern
23075  a03c
```

```
23076  a03c  c8              iny             Zeiger in Speicherseite
23077  a03d  d0 f8           bne clh020      akt. PGM verschieben
23078  a03f
23079  a03f  e6 25           inc index1+1    ADDRH neue Untergrenze
23080  a041  e6 27           inc index2+1    ADDRH alte Untergrenze
23081  a043
23082  a043  ===> Test Verschiebung fertig <===
23083  a043
23084  a043  ad 11 12        lda txttop+1    ADDRH Obergrenze akt. PGM
23085  a046  c5 27           cmp index2+1    ADDRH alte Untergrenze
23086  a048  b0 ed           bcs clh020      akt. PGM verschieben
23087  a04a
23088  a04a
23088  a04a  ===> BASIC Zeiger um 9 KB erniedrigen <===
23089  a04a
23090  a04a  38              sec
23091  a04b  a5 2e           lda txttab+1    ADDRH Untergrenze akt. PGM
23092  a04d  e9 24           sbc #>_graph    - 9 KB
23093  a04f  85 2e           sta txttab+1    ADDRH Untergrenze akt. PGM
23094  a051
23095  a051  ad 11 12        lda txttop+1    ADDRH Obergrenze akt. PGM
23096  a054  e9 24           sbc #>_graph    - 9 KB
23097  a056  8d 11 12        sta txttop+1    ADDRH Obergrenze akt. PGM
23098  a059
23099  a059  a5 44           lda datptr+1    ADDRH Zeiger auf DATA (READ)
23100  a05b  e9 24           sbc #>_graph    - 9 KB
23101  a05d  85 44           sta datptr+1    ADDRH Zeiger auf DATA (READ)
23102  a05f
23103  a05f  4c c7 9f        jmp sth080      BASIC Zeiger um 9KB erhöhen
23104  a062
23105  a062
23106  a062  ===> ADDRH aus BASIC Stapel korr. <===
23107  a062
23108  a062  b1 3f     addoff lda (fndpnt),y  ADDRL BASIC-Stapel Suchzeiger
23109  a064
23110  a064  24 76           bit mvdflg      Flag GRAPHIC-RAM
23111  a066
23112  a066   $00  kein GRAPHIC RAM eingerichtet
23113  a066   $ff  GRAPHIC RAM eingerichtet
23114  a066
23115  a066  d0 06           bne adof10      ADDRH um 9 KB erhöhen
23116  a068
23117  a068  ===> ADDRH um 9 KB erniedrigen <===
23118  a068
23119  a068  38              sec
23120  a069  e9 24           sbc #>_graph    - 9 KB
23121  a06b  91 3f           sta (fndpnt),y  ADDRL BASIC-Stapel Suchzeiger
23122  a06d  60              rts
23123  a06e
23124  a06e
23125  a06e  ===> ADDRH um 9 KB erhöhen <===
23126  a06e
23127  a06e  18       adof10 clc
23128  a06f  69 24           adc #>_graph    + 9 KB
23129  a071  91 3f           sta (fndpnt),y  ADDRL BASIC-Stapel Suchzeiger
23130  a073  60              rts
23131  a074
23132  a074
```

```
23133  a074  ===> Test GRAPHIC-RAM reserviert <===
23134  a074
23135  a074  a5 76      isgrap lda mvdflg       Flag GRAPHIC-RAM
23136  a076  f0 01             beq isgrer       GRAPHIC RAM nicht reserviert
23137  a078
23138  a078  60               rts
23139  a079
23140  a079
23141  a079  ===> Behandlung »?no graphics area« <===
23142  a079
23143  a079  a2 23      isgrer ldx #_erng
23144  a07b  4c 3c 4d         jmp error        ind. Behandlung Fehlermeldung
23145  a07e
23146  a07e
23147  a07e                  .end
23148  a07e                  .lib basic.diskb_1
23149  a07e  put"@basic.diskb_1"
```

```
23151  a07e
23152  a07e
23153  a07e  ===> Disk-B.-Statem. DIRECTORY/CATALOG ($ee/ $fe $0c) <===
23154  a07e
23155  a07e  20 bf a3   catlog jsr dospar       Param. einlesen/prüfen
23156  a081
23157  a081  a5 80             lda parsts       DOS Syntax Byte 1
23158  a083  29 e6             and #255-_fn1-_fa-_d1
23159  a085  f0 03             beq dcat0
23160  a087
23161  a087  ===> Behandlung »?syntax error« <===
23162  a087
23163  a087  4c 6c 79          jmp snerr        Behandlung »?syntax error«
23164  a08a
23165  a08a
23166  a08a  a0 01      dcat0  ldy #pdir-tabld-1
23167  a08c  a2 01             ldx #1           Länge Befehlsstring DIRECTORY
23168  a08e
23169  a08e  a5 80             lda parsts       DOS Syntax Byte 1
23170  a090  29 11             and #_d1+_fn1
23171  a092  f0 06             beq dcat2
23172  a094
23173  a094  4a                lsr a
23174  a095
23175  a095   ......1  1 = Filename 1 angegeben
23176  a095
23177  a095  90 02             bcc dcat1        CATALOG/DIRECTORY mit Lw
23178  a097
23179  a097  ===> CATALOG/DIRECTORY mit Filenamen <===
23180  a097
23181  a097  e8                inx              für »:«
23182  a098  e8                inx              für 1. Filenamen
23183  a099
23184  a099
23185  a099  ===> CATALOG/DIRECTORY mit Lw <===
23186  a099
23187  a099  e8         dcat1  inx              für 1. Laufwerk
23188  a09a
23189  a09a  8a         dcat2  txa              Länge Befehlsstring
23190  a09b  20 67 a6          jsr sendp        DOS-Befehlsstring zusammensetzen u.
                                                ausgeben
23191  a09e
23192  a09e  a9 00             lda #_pgmbk      Bank für DOS Operation
23193  a0a0  aa                tax              Bank für Filenamen
23194  a0a1  20 87 92          jsr cstbnk       ac nach RAM-Bank, xr nach
                                                Filename-Bank
23195  a0a4
23196  a0a4  a0 60             ldy #_talk+_defsa
23197  a0a6  ae 1c 01          ldx dosfa        Primäradresse
23198  a0a9  a9 00             lda #_defla
23199  a0ab  20 57 92          jsr cstlfs       ac, xr, yr nach la, fa, sa
23200  a0ae
23201  a0ae  38                sec
23202  a0af  20 d8 90          jsr copen        BASIC-Statement OPEN ($9f)
23203  a0b2  90 09             bcc dcat21       listen für directory
23204  a0b4
23205  a0b4
```

```
23205  a0b4  ===> Behandlung Fehler DIRECTORY <===
23206  a0b4
23207  a0b4  48              pha             I/O Fehlernummer
23208  a0b5
23209  a0b5  20 14 a1        jsr dcat10      la DIRECTORY schließen
23210  a0b8  68              pla
23211  a0b9  aa              tax             I/O Fehlernummer
23212  a0ba  4c 3c 4d        jmp error       ind. Behandlung Fehlermeldung
23213  a0bd
23214  a0bd
23215  a0bd  ===> listen für directory <===
23216  a0bd
23217  a0bd  a2 00     dcat21 ldx #_defla
23218  a0bf  20 45 a8        jsr mapio       I/O-Chips einblenden
23219  a0c2  20 c6 ff        jsr kchkin      Eingabekanal üb. Vector öffnen
                                             (Sprungv.)
23220  a0c5
23221  a0c5
23222  a0c5   Commodore Floppies geben die DIRECTORY als Pseudo BASIC PGM aus:
23223  a0c5
23224  a0c5   Die ersten beiden Bytes sind ADDRL/ADDRH zum Laden des PGM,
23225  a0c5
23226  a0c5   darauf folgt die erste PGM Zeile, bestehend aus:
23227  a0c5
23228  a0c5     Zeilenkopf: 2 Bytes Dummy Zeilenlink $0101,
23229  a0c5                 Zeilennummer 16 Bit INT Laufwerksnummer low/high
23230  a0c5
23231  a0c5     Steuerzeichen _rvson, Headername in Anf.z., ID und Floppy Typ
23232  a0c5     Zeilenendezeichen (_eom)
23233  a0c5
23234  a0c5   die folgenden BASIC PGM Zeilen bestehen aus:
23235  a0c5
23236  a0c5     Zeilenkopf: 2 Bytes Dummy Zeilenlink $0101
23237  a0c5                 Zeilennummer 16 Bit INT Anzahl belegte Blöcke
23238  a0c5
23239  a0c5     PGM Name in Anführungszeichen, File Type (SEQ, PRG, USR, REL)
23240  a0c5     Zeilenendezeichen (_eom)
23241  a0c5
23242  a0c5   die letzte BASIC PGM Zeile besteht aus:
23243  a0c5
23244  a0c5     Zeilenkopf: 2 Bytes Dummy Zeilenlink $0101
23245  a0c5                 Zeilennummer 16 Bit INT Anzahl freie Blöcke
23246  a0c5
23247  a0c5     Text »blocks free.«
23248  a0c5     Zeilenendezeichen (_eom)
23249  a0c5
23250  a0c5
23251  a0c5  a0 03           ldy #2+_hedln/2
23252  a0c7
23253  a0c7
23254  a0c7  ===> Linkadresse, Zeilennummer lesen <===
23255  a0c7
23256  a0c7  8c 74 11  dcat22 sty t3          Zähler
23257  a0ca
23258  a0ca  20 63 92  dcat23 jsr cbasin      ind. Zeich. akt. Kanal nach ac
23259  a0cd  8d 75 11        sta t4          Low-Byte 16 Bit INT
23260  a0d0
23261  a0d0  20 51 92        jsr creads      RS-/System-Status nach ac
```

```
23262  a0d3  d0 3f           bne dcat10       la DIRECTORY schließen
23263  a0d5
23264  a0d5  20 63 92        jsr cbasin       ind. Zeich. akt. Kanal nach ac
23265  a0d8  8d 76 11        sta t5           High-Byte 16 Bit INT
23266  a0db
23267  a0db  20 51 92        jsr creads       RS-/System-Status nach ac
23268  a0de  d0 34           bne dcat10       la DIRECTORY schließen
23269  a0e0
23270  a0e0  ce 74 11        dec t3
23271  a0e3  d0 e5           bne dcat23
23272  a0e5
23273  a0e5  ae 75 11        ldx t4           Low-Byte 16 Bit INT
23274  a0e8  ad 76 11        lda t5           High-Byte 16 Bit INT
23275  a0eb  20 32 8e        jsr linprt       xr/ac von hex nach dez, Ausgabe in
                                              ASCII
23276  a0ee
23277  a0ee  a9 20           lda #_spc
23278  a0f0  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
23279  a0f3
23280  a0f3
23281  a0f3  ===> DIRECTORY Zeile lesen/ausgeben <===
23282  a0f3
23283  a0f3  20 63 92  dcat4 jsr cbasin       Zeichen aus Zeile nach ac
23284  a0f6  48               pha              Zeichen
23285  a0f7
23286  a0f7  20 51 92        jsr creads       RS-/System-Status nach ac
23287  a0fa  d0 17           bne dcat9        Zeichen verwerfen, Kanal schließen
23288  a0fc
23289  a0fc  68              pla              Zeichen
23290  a0fd  f0 06           beq dcat5        Zeilenende, Stopp Test, nächste
                                              Zeile
23291  a0ff
23292  a0ff  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
23293  a102  4c f3 a0        jmp dcat4        DIRECTORY Zeile lesen/ausgeben
23294  a105
23295  a105
23296  a105  ===> Zeilenende, Stopp Test, nächste Zeile <===
23297  a105
23298  a105  a9 0d      dcat5 lda #_cr
23299  a107  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
23300  a10a
23301  a10a  20 93 92        jsr cstop        Abfrage Stopp-Taste, dann Ersatz-I/O
23302  a10d  f0 05           beq dcat10       la DIRECTORY schließen
23303  a10f
23304  a10f  a0 02           ldy #_hedln/2
23305  a111  d0 b4           bne dcat22       Linkadresse, Zeilennummer lesen
23306  a113
23307  a113
23308  a113  ===> Zeichen verwerfen, Kanal schließen <===
23309  a113
23310  a113  68          dcat9 pla
23311  a114
23312  a114
23313  a114  ===> la DIRECTORY schließen <===
23314  a114
23315  a114  20 6f 92   dcat10 jsr cclrch      ind. I/O-Kanäle schließen
23316  a117
23317  a117  a9 00           lda #_defla
```

```
23318  a119  18                clc
23319  a11a  4c 75 92          jmp cclose        log. File (la in ac) schließen
23320  a11d
23321  a11d
23322  a11d  ===> Disk-B.-Statem. DOPEN ($fe $0d) <===
23323  a11d
23324  a11d  a9 22       dopen   lda #_d2+_fn2    nicht erlaubt
23325  a11f  20 c1 a3            jsr dosprs        Param. einlesen/mit ac prüfen
23326  a122
23327  a122  20 6f a7            jsr chk6          Syntaxprüfung APPEND, DOPEN
23328  a125  20 57 a1            jsr fndsca        freie sa suchen/eintragen
23329  a128
23330  a128  a0 05               ldy #pdop-tabld-1
23331  a12a  a2 04               ldx #4            Länge Befehlsstring DOPEN
23332  a12c
23333  a12c  24 80               bit parsts        DOS Syntax Byte 1
23334  a12e
23335  a12e   .6.....  1 = dosrcl eingetragen
23336  a12e
23337  a12e  50 13               bvc dop2
23338  a130
23339  a130  a2 08               ldx #8            Länge Befehlsstr. DOPEN mit dosrcl
23340  a132  d0 0f               bne dop2
23341  a134
23342  a134
23343  a134  ===> Disk-B.-Statem. APPEND ($fe $0e) <===
23344  a134
23345  a134  a9 e2       append  lda #_rpl+_rcl+_d2+_fn2 nicht erlaubt
23346  a136  20 c1 a3            jsr dosprs        Param. einlesen/mit ac prüfen
23347  a139
23348  a139  20 6f a7            jsr chk6          Syntaxprüfung APPEND, DOPEN
23349  a13c  20 57 a1            jsr fndsca        freie sa suchen/eintragen
23350  a13f
23351  a13f  a0 16               ldy #papp-tabld-1
23352  a141  a2 05               ldx #5            Länge Befehlsstring APPEND
23353  a143
23354  a143  8a         dop2     txa               Länge Befehlsstring
23355  a144  20 67 a6            jsr sendp         DOS-Befehlsstring zusammensetzen u.
                                                   ausgeben
23356  a147  20 6f 92            jsr cclrch        ind. I/O-Kanäle schließen
23357  a14a
23358  a14a  a9 00               lda #_pgmbk       Bank für D.-BASIC
23359  a14c  aa                  tax               Bank Filenamen
23360  a14d  20 87 92            jsr cstbnk        ac nach RAM-Bank, xr nach
                                                   Filename-Bank
23361  a150
23362  a150  20 d8 90            jsr copen         BASIC-Statement OPEN ($9f)
23363  a153
23364  a153  38                  sec
23365  a154  4c 75 92            jmp cclose        log. File (la in ac) schließen
23366  a157
23367  a157
23368  a157  ===> freie sa suchen/eintragen <===
23369  a157
23370  a157  a0 61       fndsca  ldy #_talk+1
23371  a159  c8          fsca10  iny
23372  a15a  c0 6f               cpy #_talk+15
23373  a15c  f0 0c               beq errtmf        Behandlung »?too many files error«
```

```
23374  a15e
23375  a15e  20 45 a8          jsr mapio        I/O-Chips einblenden
23376  a161  20 5c ff          jsr lkups        Geräte-Adr. über Sek.Adr. (yr)
                                                 suchen (Sprungv.)
23377  a164  90 f3             bcc fsca10
23378  a166
23379  a166  8c 1d 01          sty dossa        Sekundäradresse
23380  a169
23381  a169  60                rts
23382  a16a
23383  a16a
23384  a16a  ===> Behandlung »?too many files error« <===
23385  a16a
23386  a16a  a2 01      errtmf ldx #_ertmf
23387  a16c  4c 3c 4d          jmp error        ind. Behandlung Fehlermeldung
23388  a16f
23389  a16f
23390  a16f  ===> Disk-B.-Statem. DCLOSE ($fe $0f) <===
23391  a16f
23392  a16f  a9 f3       dclose lda #255-_fa-_la erlaubt
23393  a171  20 c1 a3          jsr dosprs       Param. einlesen/mit ac prüfen
23394  a174
23395  a174  20 0d a8          jsr oldclr       DS$-String ungültig machen
23396  a177
23397  a177  ===> Test auf la, sonst alle Files schließen <===
23398  a177
23399  a177  a5 80             lda parsts       DOS Syntax Byte 1
23400  a179  29 04             and #_la
23401  a17b  f0 06             beq dclall       alle logischen Files schließen
23402  a17d
23403  a17d  ad 1b 01          lda dosla        logische Adresse
23404  a180  4c 75 92          jmp cclose       log. File (la in ac) schließen
23405  a183
23406  a183
23407  a183  ===> alle logischen Files schließen <===
23408  a183
23409  a183  ad 1c 01   dclall lda dosfa        Primäradresse
23410  a186  20 45 a8          jsr mapio        I/O-Chips einblenden
23411  a189  4c 4a ff          jmp kclsal       alle Files von Gerät in ac schließen
                                                 (Sprungv.)
23412  a18c
23413  a18c
23414  a18c  ===> Disk-B.-Statem. DSAVE ($ef) <===
23415  a18c
23416  a18c  a9 66       dsave  lda #255-_rpl-_d1-_fa-_fn1 erlaubt
23417  a18e  20 c1 a3          jsr dosprs       Param. einlesen/mit ac prüfen
23418  a191
23419  a191  20 50 a7          jsr chk2         Syntaxprüfung DLOAD, DSAVE
23420  a194
23421  a194  a0 05             ldy #pdop-tabld-1
23422  a196  a9 04             lda #4           Länge Befehlsstring DSAVE
23423  a198  20 67 a6          jsr sendp        DOS-Befehlsstring zusammensetzen u.
                                                 ausgeben
23424  a19b
23425  a19b  a9 00             lda #_pgmbk      Bank für D.-BASIC
23426  a19d  aa                tax              Bank Filenamen
23427  a19e  20 87 92          jsr cstbnk       ac nach RAM-Bank, xr nach
                                                 Filename-Bank
```

```
23428  a1a1
23429  a1a1  4c 15 91          jmp savenp       SAVE, Param. eingelesen
23430  a1a4
23431  a1a4
23432  a1a4
23432  a1a4  ===> Disk-B.-Statem. DVERIFY ($fe $14) <===
23433  a1a4
23434  a1a4  a9 01      dverfy lda #1           Flag für VERIFY
23435  a1a6  2c                .byte _skip2
23436  a1a7
23437  a1a7
23438  a1a7  ===> Disk-B.-Statem. DLOAD ($f0) <===
23439  a1a7
23440  a1a7  a9 00      dload  lda #0           Flag für LOAD
23441  a1a9  85 0c             sta bverck       Flag LOAD=0, VERIFY=1 (BASIC)
23442  a1ab
23443  a1ab  a9 e6             lda #255-_d1-_fa-_fn1  erlaubt
23444  a1ad  20 c1 a3          jsr dosprs       Param. einlesen/mit ac prüfen
23445  a1b0
23446  a1b0  20 50 a7          jsr chk2         Syntaxprüfung DLOAD, DSAVE
23447  a1b3
23448  a1b3  a9 00             lda #_defsa
23449  a1b5  8d 1d 01          sta dossa        Sekundäradresse
23450  a1b8
23451  a1b8  a0 05             ldy #pdop-tabld-1
23452  a1ba  a9 04             lda #4           Länge Befehlsstring DVERIFY/DLOAD
23453  a1bc  20 67 a6          jsr sendp        DOS-Befehlsstring zusammensetzen u.
                                                 ausgeben
23454  a1bf
23455  a1bf  a9 00             lda #_pgmbk      Bank für D.-BASIC
23456  a1c1  aa                tax              Bank Filenamen
23457  a1c2  20 87 92          jsr cstbnk       ac nach RAM-Bank, xr nach
                                                 Filename-Bank
23458  a1c5
23459  a1c5  4c 33 91          jmp loadnp       LOAD/VERIFY, Param. eingelesen
23460  a1c8
23461  a1c8
23462  a1c8  ===> Disk-B.-Statem. BSAVE ($fe $10) <===
23463  a1c8
23464  a1c8  a9 66      bsave  lda #255-_rpl-_d1-_fa-_fn1 erlaubt
23465  a1ca  a2 f8             ldx #255-_bnk-_stadr-_endad erlaubt
23466  a1cc  20 c3 a3          jsr dosprx       Param. einlesen/mit ac/xr prüfen
23467  a1cf
23468  a1cf  20 50 a7          jsr chk2         Syntaxprüfung DLOAD, DSAVE
23469  a1d2
23470  a1d2  ===> Test Eingabe Start- und Endadresse <===
23471  a1d2
23472  a1d2  a5 81             lda parstx       DOS Syntax Byte 2
23473  a1d4  29 06             and #_endad+_stadr
23474  a1d6  c9 06             cmp #_endad+_stadr
23475  a1d8  f0 03             beq bsav1
23476  a1da
23477  a1da  ===> Behandlung »?syntax error« <===
23478  a1da
23479  a1da  4c 6c 79          jmp snerr        Behandlung »?syntax error«
23480  a1dd
23481  a1dd
23482  a1dd  ad 1a 01   bsav1  lda dosofh+1      ADDRH Obergrenze BLOAD/BSAVE
```

```
23483  a1e0  cd 18 01          cmp dosofl+1      ADDRH Untergrenze BLOAD/BSAVE
23484  a1e3  90 30             bcc qtyer3        Behandlung »?illegal quantity error«
23485  a1e5  d0 0a             bne bsav2
23486  a1e7
23487  a1e7  ad 19 01          lda dosofh        ADDRL Obergrenze BLOAD/BSAVE
23488  a1ea  cd 17 01          cmp dosofl        ADDRL Untergrenze BLOAD/BSAVE
23489  a1ed  90 26             bcc qtyer3        Behandlung »?illegal quantity error«
23490  a1ef  f0 24             beq qtyer3        Behandlung »?illegal quantity error«
23491  a1f1
23492  a1f1  a0 05       bsav2 ldy #pdop-tabld-1
23493  a1f3  a9 04             lda #4            Länge Befehlsstring BSAVE
23494  a1f5  20 67 a6          jsr sendp         DOS-Befehlsstring zusammensetzen u.
                                                 ausgeben
23495  a1f8
23496  a1f8  ad 1f 01          lda dosbnk        Bank für D.-BASIC
23497  a1fb  a2 00             ldx #_pgmbk       Bank Filename
23498  a1fd  20 87 92          jsr cstbnk        ac nach RAM-Bank, xr nach
                                                 Filename-Bank
23499  a200
23500  a200  ae 17 01          ldx dosofl        ADDRL Untergrenze BLOAD/BSAVE
23501  a203  ac 18 01          ldy dosofl+1      ADDRH Untergrenze BLOAD/BSAVE
23502  a206
23503  a206  a9 5a             lda #<highds      Zeiger auf Untergrenze
23504  a208
23505  a208  86 5a             stx highds        ADDRL Untergrenze BLOAD/BSAVE
23506  a20a  84 5b             sty highds+1      ADDRH Untergrenze BLOAD/BSAVE
23507  a20c
23508  a20c  ae 19 01          ldx dosofh        ADDRL Obergrenze BLOAD/BSAVE
23509  a20f  ac 1a 01          ldy dosofh+1      ADDRH Obergrenze BLOAD/BSAVE
23510  a212
23511  a212  4c 1d 91          jmp savenb        Teil der save-Routine
23512  a215
23513  a215
23514  a215  ===> Behandlung »?illegal quantity error« <===
23515  a215
23516  a215  4c 28 7d   qtyer3 jmp fcerr        Behandlung »?illegal quantity error«
23517  a218
23518  a218
23519  a218  ===> Disk-B.-Statem. BLOAD ($fe $11) <===
23520  a218
23521  a218  a9 e6       bload  lda #255-_d1-_fa-_fn1  erlaubt
23522  a21a  a2 fc             ldx #255-_stadr-_bnk   erlaubt
23523  a21c  20 c3 a3          jsr dosprx        Param. einlesen/mit ac/xr prüfen
23524  a21f
23525  a21f
23526  a21f  ===> BLOAD, Param. eingelesen <===
23527  a21f
23528  a21f  20 50 a7   blodnp jsr chk2          Syntaxprüfung DLOAD, DSAVE
23529  a222
23530  a222  ae 17 01          ldx dosofl        ADDRL Untergrenze BLOAD/BSAVE
23531  a225  ac 18 01          ldy dosofl+1      ADDRH Untergrenze BLOAD/BSAVE
23532  a228
23533  a228  a9 00             lda #$00
23534  a22a  e0 ff             cpx #$ff
23535  a22c  d0 06             bne bload2
23536  a22e
23537  a22e  c0 ff             cpy #$ff
23538  a230  d0 02             bne bload2
```

```
23539  a232
23540  a232  a9 ff           lda #$ff
23541  a234  8d 1d 01  bload2 sta dossa        Sekundäradresse
23542  a237
23543  a237  a0 05           ldy #pdop-tabld-1
23544  a239  a9 04           lda #4            Länge Befehlsstring BLOAD
23545  a23b  20 67 a6        jsr sendp         DOS-Befehlsstring zusammensetzen u.
                                               ausgeben
23546  a23e
23547  a23e  ad 1f 01        lda dosbnk        Bank für D.-BASIC
23548  a241  a2 00           ldx #_pgmbk       Bank Filename
23549  a243  20 87 92        jsr cstbnk        ac nach RAM-Bank, xr nach
                                               Filename-Bank
23550  a246
23551  a246  a9 00           lda #0            LOAD Flag
23552  a248  ae 17 01        ldx dosofl        ADDRL Untergrenze BLOAD/BSAVE
23553  a24b  ac 18 01        ldy dosofl+1      ADDRH Untergrenze BLOAD/BSAVE
23554  a24e  20 d5 ff        jsr kloads        load/verify (ac) ab Adresse in xr/yr
                                               (Sprungv.)
23555  a251
23556  a251  08              php
23557  a252  20 43 92        jsr dschk         für FA > 3 DS$ rücksetzen
23558  a255  28              plp
23559  a256  90 03           bcc bload3
23560  a258
23561  a258  ===> Fehlerbehandlung BLOAD <===
23562  a258
23563  a258  4c d0 90        jmp erexit        Behandlung I/O-Fehler
23564  a25b
23565  a25b
23566  a25b  20 51 92  bload3 jsr creads        RS-/System-Status nach ac
23567  a25e  29 bf           and #%10111111
23568  a260
23569  a260   .6.....  1 = End of File
23570  a260
23571  a260  f0 03           beq bload4
23572  a262
23573  a262  ===> Fehlerbehandlung BLOAD <===
23574  a262
23575  a262  4c 67 91        jmp loderr        Behandlung »?load error«
23576  a265
23577  a265
23578  a265  18        bload4 clc
23579  a266  60               rts
23580  a267
23581  a267
23582  a267  ===> Disk-B.-Statem. HEADER ($f1) <===
23583  a267
23584  a267  20 bf a3   header jsr dospar       Param. einlesen/prüfen
23585  a26a
23586  a26a  20 49 a7         jsr chk1          Syntaxprüfung HEADER, SCRATCH
23587  a26d
23588  a26d  ===> Test auf Headername <===
23589  a26d
23590  a26d  29 01           and #_fn1
23591  a26f  c9 01           cmp #_fn1
23592  a271  d0 61           bne sner1         Headername fehlt
23593  a273
```

```
23594  a273  20 7b 92          jsr cclall       off. Files :=0, Ersatz-I/O
23595  a276  20 e1 a7          jsr rusure       Bearbeitung »are you sure ?«
23596  a279  d0 25             bne bsvrts       Antwort nicht »y«
23597  a27b
23598  a27b  a0 1b             ldy #phed-tabld-1
23599  a27d  a9 04             lda #4           Länge Befehlsstring HEADER ohne ID
23600  a27f  ae 20 01          ldx dosdid       ID erstes Zeichen
23601  a282  f0 02             beq frmt6        HEADER ohne ID
23602  a284
23603  a284  a9 06             lda #6           Länge Befehlsstring HEADER mit ID
23604  a286
23605  a286  20 97 a3   frmt6  jsr trans        Befehlsstring bauen, ausgeben,
                                                listen, open
23606  a289  20 78 a7          jsr errchl       DS$ über Sek.adr. 15 einlesen
23607  a28c
23608  a28c  24 7f             bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
23609  a28e  30 10             bmi bsvrts       PGM Modus
23610  a290
23611  a290  a0 00             ldy #0           Zeiger in DS$-String
23612  a292  a9 7b             lda #<dsdesc+1
23613  a294  20 ab 03          jsr iram1        Zeichen DS$ nach ac
23614  a297  c9 32             cmp #'2'         darf 0 oder 1 sein
23615  a299  90 05             bcc bsvrts       ok
23616  a29b
23617  a29b  ===> Behandlung »?bad disk error« <===
23618  a29b
23619  a29b  a2 24             ldx #_erbdk
23620  a29d  4c 3c 4d          jmp error        ind. Behandlung Fehlermeldung
23621  a2a0
23622  a2a0
23623  a2a0  60         bsvrts rts
23624  a2a1
23625  a2a1
23626  a2a1  ===> Disk-B.-Statem. SCRATCH ($f2) <===
23627  a2a1
23628  a2a1  20 bf a3   scratc jsr dospar       Param. einlesen/prüfen
23629  a2a4  20 49 a7          jsr chk1         Syntaxprüfung HEADER, SCRATCH
23630  a2a7
23631  a2a7  20 e1 a7          jsr rusure       Bearbeitung »are you sure ?«
23632  a2aa  d0 27             bne zxit         Antwort nicht »y«
23633  a2ac
23634  a2ac  a0 37             ldy #pscr-tabld-1
23635  a2ae  a9 04             lda #4           Länge Befehlsstring SCRATCH
23636  a2b0  20 97 a3          jsr trans        Befehlsstring bauen, ausgeben,
                                                listen, open
23637  a2b3
23638  a2b3  20 78 a7          jsr errchl       DS$ über Sek.adr. 15 einlesen
23639  a2b6
23640  a2b6  24 7f             bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
23641  a2b8  30 19             bmi zxit         PGM Modus
23642  a2ba
23643  a2ba  a9 0d             lda #_cr
23644  a2bc  20 69 92          jsr cbsout       ind. Ausgabe ac auf akt. Kanal
23645  a2bf
23646  a2bf  a0 00             ldy #0           Zeiger in DS$-String
23647  a2c1
23648  a2c1  a9 7b      sctc1  lda #<dsdesc+1
23649  a2c3  20 ab 03          jsr iram1        Zeichen aus DS$ nach ac
```

```
23650  a2c6  f0 06           beq sctc2        _eom
23651  a2c8
23652  a2c8  20 69 92        jsr cbsout       ind. Ausgabe ac auf akt. Kanal
23653  a2cb  c8              iny              Zeiger in DS$-String
23654  a2cc  d0 f3           bne sctc1        immer
23655  a2ce
23656  a2ce
23657  a2ce  a9 0d     sctc2 lda #_cr
23658  a2d0  20 df 90        jsr outch        Ausgabe ac akt. Kanal, Fehlertest
                                              (BASIC)
23659  a2d3
23660  a2d3  60        zxit  rts
23661  a2d4
23662  a2d4
23663  a2d4  ===> Behandlung »?syntax error« <===
23664  a2d4
23665  a2d4  4c 6c 79  sner1 jmp snerr        Behandlung »?syntax error«
23666  a2d7
23667  a2d7
23668  a2d7  ===> Disk-B.-Statem. RECORD ($fe $12) <===
23669  a2d7
23670  a2d7  a9 23     record lda #'#'
23671  a2d9  20 5e 79        jsr synchr       Test auf Zeichen in ac, sonst Fehler
23672  a2dc
23673  a2dc  20 f4 87        jsr getbyt       la nach xr
23674  a2df  e0 00           cpx #_defla
23675  a2e1  f0 37           beq fcer         Behandlung »?illegal quantity error«
23676  a2e3
23677  a2e3  8e 1b 01        stx dosla        logische Adresse
23678  a2e6
23679  a2e6  20 0f 88        jsr comwrd       Rec.Nr. als 16-Bit INT nach linnum
23680  a2e9
23681  a2e9  a2 01           ldx #1           Vorgabe Record Länge
23682  a2eb  20 1e 9e        jsr optbyt       Record Länge opt. nach xr
23683  a2ee
23684  a2ee  e0 00           cpx #0
23685  a2f0  f0 28           beq fcer         Behandlung »?illegal quantity error«
23686  a2f2
23687  a2f2  e0 ff           cpx #255
23688  a2f4  f0 24           beq fcer         Behandlung »?illegal quantity error«
23689  a2f6
23690  a2f6  8e 1e 01        stx dosrcl       Record Länge
23691  a2f9
23692  a2f9  ad 1b 01        lda dosla        logische Adresse
23693  a2fc  20 45 a8        jsr mapio        I/O-Chips einblenden
23694  a2ff  20 59 ff        jsr klkupl       Geräte- & Sek.-Adr. über la (ac)
                                              suchen (Sprungv.)
23695  a302  b0 19           bcs fnoerr       Behandlung »?file not open error«
23696  a304
23697  a304  8c ed 11        sty tdossa
23698  a307  8e 1c 01        stx dosfa        Primäradresse
23699  a30a
23700  a30a  a9 00           lda #_defla
23701  a30c  8d 1b 01        sta dosla        logische Adresse
23702  a30f
23703  a30f  a9 6f           lda #_talk+_cmdsa
23704  a311  8d 1d 01        sta dossa        Sekundäradresse
23705  a314
```

```
23706  a314  a0 3b            ldy #prec-tabld-1
23707  a316  a9 04            lda #4          Länge Befehlsstring RECORD
23708  a318  d0 7d            bne trans       Befehlsstring bauen, ausgeben,
                                               listen, open
23709  a31a
23710  a31a
23711  a31a  ===> Behandlung »?illegal quantity error« <===
23712  a31a
23713  a31a  4c 28 7d   fcer   jmp fcerr       Behandlung »?illegal quantity error«
23714  a31d
23715  a31d
23716  a31d  ===> Behandlung »?file not open error« <===
23717  a31d
23718  a31d  a2 04      fnoerr ldx #_erfnf
23719  a31f  4c 3c 4d          jmp error       ind. Behandlung Fehlermeldung
23720  a322
23721  a322
23722  a322  ===> Disk-B.-Statem. DCLEAR ($fe $15) <===
23723  a322
23724  a322  20 bf a3   dclear jsr dospar      Param. einlesen/prüfen
23725  a325  a0 ff            ldy #$ff        pdclr-tabld-1
23726  a327  a9 02            lda #2          Länge Befehlsstring DCLEAR
23727  a329  20 97 a3         jsr trans       Befehlsstring bauen, ausgeben,
                                               listen, open
23728  a32c
23729  a32c  4c 83 a1         jmp dclall      alle logischen Files schließen
23730  a32f
23731  a32f
23732  a32f  ===> Disk-B.-Statem. COLLECT ($f3) <===
23733  a32f
23734  a32f  20 bf a3   colect jsr dospar      Param. einlesen/prüfen
23735  a332  20 5b a7         jsr chk3        Syntaxprüfung CATALOG, COLLECT
23736  a335  20 7b 92         jsr cclall      off. Files :=0, Ersatz-I/O
23737  a338
23738  a338  a0 21            ldy #pcoll-tabld-1
23739  a33a  a2 01            ldx #1          Länge Befehlsstring COLLECT ohne Lw
23740  a33c
23741  a33c  ===> Test auf Angabe Laufwerk <===
23742  a33c
23743  a33c  a5 80            lda parsts      DOS Syntax Byte 1
23744  a33e  29 10            and #_d1
23745  a340  f0 01            beq clct2       COLLECT ohne Lw
23746  a342
23747  a342  e8               inx             Länge Befehlsstring COLLECT mit Lw
23748  a343
23749  a343  8a         clct2 txa             Länge Befehlsstring
23750  a344  d0 51            bne trans       immer
23751  a346
23752  a346
23753  a346  ===> Disk-B.-Statem. COPY ($f4) <===
23754  a346
23755  a346  20 bf a3   dcopy  jsr dospar      Param. einlesen/prüfen
23756  a349  29 30            and #_d1+_d2
23757  a34b  c9 30            cmp #_d1+_d2
23758  a34d  d0 06            bne dcpy2       nur 1 Lw angegeben
23759  a34f
```

```
23760  a34f  ===> Syntaxtest COPY ohne Filenamen <===
23761  a34f
23762  a34f  a5 80           lda parsts       DOS Syntax Byte 1
23763  a351  29 c7           and #255-_d2-_d1-_fa   erlaubt
23764  a353  f0 07           beq dcpy4        ok
23765  a355
23766  a355
23767  a355  ===> Syntaxtest COPY mit Filenamen <===
23768  a355
23769  a355  a5 80     dcpy2 lda parsts       DOS Syntax Byte 1
23770  a357  20 60 a7         jsr chk4         Syntaxprüfung CONCAT, COPY
23771  a35a
23772  a35a  a5 80           lda parsts       ************* BUG ************
23773  a35c
23774  a35c  a0 27     dcpy4 ldy #pdcop-tabld-1
23775  a35e  a9 08           lda #8           Länge Befehlsstring COPY
23776  a360  d0 35           bne trans        immer
23777  a362
23778  a362
23779  a362  ===> Disk-B.-Statem. CONCAT ($fe $13) <===
23780  a362
23781  a362  20 bf a3  concat jsr dospar      Param. einlesen/prüfen
23782  a365
23783  a365  20 60 a7         jsr chk4         Syntaxprüfung CONCAT, COPY
23784  a368
23785  a368  a0 0d           ldy #pconc-tabld-1
23786  a36a  a9 0c           lda #12          Länge Befehlsstring CONCAT
23787  a36c  d0 29           bne trans        immer
23788  a36e
23789  a36e
23790  a36e  ===> Disk-B.-Statem. RENAME ($f5) <===
23791  a36e
23792  a36e  a9 e4     rename lda #255-_d1-_fa-_fn2-_fn1 erlaubt
23793  a370  20 c1 a3         jsr dosprs       Param. einlesen/mit ac prüfen
23794  a373
23795  a373  20 66 a7         jsr chk5         Syntaxprüfung RENAME
23796  a376
23797  a376  a0 2f           ldy #pren-tabld-1
23798  a378  a9 08           lda #8           Länge Befehlsstring RENAME
23799  a37a  d0 1b           bne trans        immer
23800  a37c
23801  a37c
23802  a37c  ===> Disk-B.-Statem. BACKUP ($f6) <===
23803  a37c
23804  a37c  a9 c7     backup lda #255-_d2-_d1-_fa   erlaubt
23805  a37e  20 c1 a3         jsr dosprs       Param. einlesen/mit ac prüfen
23806  a381
23807  a381  29 30           and #_d2+_d1
23808  a383  c9 30           cmp #_d2+_d1
23809  a385  f0 03           beq bup0         zuwenig Lw
23810  a387
23811  a387  ===> Behandlung »?syntax error« <===
23812  a387
23813  a387  4c 6c 79         jmp snerr        Behandlung »?syntax error«
23814  a38a
23815  a38a
23816  a38a  20 e1 a7  bup0   jsr rusure       Bearbeitung »are you sure ?«
23817  a38d  f0 01           beq bup1         Antwort »y«
```

```
23818  a38f
23819  a38f  60                rts
23820  a390
23821  a390
23822  a390  20 83 a1   bup1   jsr dclall       alle logischen Files schließen
23823  a393
23824  a393  a0 23             ldy #pbak-tabld-1
23825  a395  a9 04             lda #4           Länge Befehlsstring BACKUP
23826  a397
23827  a397
23828  a397  ===> Befehlsstring bauen, ausgeben, listen, open <===
23829  a397
23830  a397  20 67 a6   trans  jsr sendp       DOS-Befehlsstring zusammensetzen u.
                                               ausgeben
23831  a39a  20 6f 92          jsr cclrch       ind. I/O-Kanäle schließen
23832  a39d
23833  a39d  a9 00             lda #_pgmbk      Bank für D.-BASIC
23834  a39f  aa                tax              Bank für Filenamen
23835  a3a0  20 87 92          jsr cstbnk       ac nach RAM-Bank, xr nach
                                               Filename-Bank
23836  a3a3
23837  a3a3  38                sec
23838  a3a4  20 d8 90          jsr copen        BASIC-Statement OPEN ($9f)
23839  a3a7
23840  a3a7  08                php
23841  a3a8  48                pha
23842  a3a9
23843  a3a9  ad 1b 01          lda dosla        logische Adresse
23844  a3ac  38                sec
23845  a3ad  20 75 92          jsr cclose       log. File (la in ac) schließen
23846  a3b0
23847  a3b0  68                pla
23848  a3b1  28                plp
23849  a3b2
23850  a3b2  b0 01             bcs ioerr3       Behandlung I/O Fehler
23851  a3b4
23852  a3b4  60                rts
23853  a3b5
23854  a3b5
23855  a3b5  4c d0 90   ioerr3 jmp erexit       Behandlung I/O Fehler
23856  a3b8
23857  a3b8
23858  a3b8  ===> Ersatzparameter für Disk-Befehl <===
23859  a3b8
23860  a3b8  ff ff      dostbl .word $ffff      Vorbesetzung Untergrenze
23861  a3ba  ff ff             .word $ffff      Vorbesetzung Obergrenze
23862  a3bc
23863  a3bc  ===> Vorbesetzung la, fa, sa <===
23864  a3bc
23865  a3bc  00                .byte _defla
23866  a3bd  08                .byte _deffa
23867  a3be  6f                .byte _talk+_cmdsa
23868  a3bf
23869  a3bf
23870  a3bf  ===> Param. einlesen/prüfen <===
23871  a3bf
23872  a3bf  a9 00      dospar lda #%00000000
23873  a3c1
```

```
23874  a3c1   7.......  0 = RPL erlaubt (@)
23875  a3c1   .6......  0 = RCL erlaubt ('l' oder ',s,w')
23876  a3c1   ..5.....  0 = d2  erlaubt (2. Filename)
23877  a3c1   ...4....  0 = d1  erlaubt (1. Filename)
23878  a3c1   ....3...  0 = fa  erlaubt (Floppy Geräteadresse)
23879  a3c1   .....2..  0 = la  erlaubt (Log. Adresse)
23880  a3c1   ......1.  0 = fn2 erlaubt (2. Filename)
23881  a3c1   .......0  0 = fn1 erlaubt (2. Filename)
23882  a3c1
23883  a3c1
23884  a3c1  ===> Param. einlesen/mit ac prüfen <===
23885  a3c1
23886  a3c1  a2 ff      dosprs ldx #%11111111
23887  a3c3
23888  a3c3   76543...      unbenutzt
23889  a3c3   .....2..  1 = ENDAD verboten (Endadresse)
23890  a3c3   ......1.  1 = STADR verboten (Startadresse)
23891  a3c3   .......0  1 = BNK   verboten (Bank)
23892  a3c3
23893  a3c3
23894  a3c3  ===> Param. einlesen/mit ac/xr prüfen <===
23895  a3c3
23896  a3c3  48         dosprx pha              Maske parsts
23897  a3c4
23898  a3c4  8a                txa
23899  a3c5  48                pha              Maske parstx
23900  a3c6
23901  a3c6  a9 00             lda #%00000000   alle Syntax Bits rücksetzen
23902  a3c8  85 80             sta parsts       DOS Syntax Byte 1
23903  a3ca  85 81             sta parstx       DOS Syntax Byte 2
23904  a3cc
23905  a3cc  ===> Puffer Befehlsstring löschen <===
23906  a3cc
23907  a3cc  a2 22             ldx #34
23908  a3ce  9d 00 01   dos01  sta fbuffr,x
23909  a3d1  ca                dex
23910  a3d2  d0 fa             bne dos01
23911  a3d4
23912  a3d4  ===> Parameter für Befehlsstring vorbesetzen <===
23913  a3d4
23914  a3d4  a2 06             ldx #6
23915  a3d6  bd b8 a3   dos02  lda dostbl,x     Ersatzparameter für Disk-Befehl
23916  a3d9  9d 17 01          sta dosofl,x
23917  a3dc  ca                dex
23918  a3dd  10 f7             bpl dos02
23919  a3df
23920  a3df  ===> DOS Bank mit akt. Bank vorbesetzen <===
23921  a3df
23922  a3df  ae d5 03          ldx curbnk       akt. Bank
23923  a3e2  8e 1f 01          stx dosbnk       Bank Nummer
23924  a3e5
23925  a3e5  20 86 03          jsr chrgot       letztes Zeichen erneut nach ac
23926  a3e8  d0 0e             bne parse1       Disk-Parameter 1 einlesen
23927  a3ea
23928  a3ea
23929  a3ea  ===> Parameter eingelesen, prüfen <===
23930  a3ea
23931  a3ea  68         done   pla              Maske parstx
```

```
23932  a3eb  25 81          and parstx      DOS Syntax Byte 2
23933  a3ed  d0 6b          bne sner2       Behandlung »?syntax error«
23934  a3ef
23935  a3ef  68             pla             Maske parsts
23936  a3f0  20 1d a6       jsr prmrpt      snerr, wenn ac-Bits parsts ges.
23937  a3f3
23938  a3f3  a5 80          lda parsts      DOS Syntax Byte 1
23939  a3f5  a6 81          ldx parstx      DOS Syntax Byte 2
23940  a3f7
23941  a3f7  60             rts
23942  a3f8
23943  a3f8
23944  a3f8                 .end
23945  a3f8                 .lib basic.diskb_2
23946  a3f8  put"@basic.diskb_2"
```

```
23948  a3f8
23949  a3f8
23950  a3f8  ===> Disk-Parameter 1 einlesen <===
23951  a3f8
23952  a3f8  c9 23      parse1 cmp #'#'
23953  a3fa  f0 4b             beq logadr       la einlesen
23954  a3fc
23955  a3fc  c9 57             cmp #'w'
23956  a3fe  f0 5d             beq reclen       Record Länge/»,s,w« einlesen
23957  a400
23958  a400  c9 4c             cmp #'l'
23959  a402  f0 59             beq reclen       Record Länge/»,s,w« einlesen
23960  a404
23961  a404  c9 52             cmp #'r'
23962  a406  f0 29             beq godel1       »r« überlesen (Ersatzwert)
23963  a408
23964  a408  c9 44             cmp #'d'
23965  a40a  f0 73             beq drv1         1. Laufwerk einlesen
23966  a40c
23967  a40c  c9 91             cmp #_ont        ON Token
23968  a40e  f0 27             beq on1          fa 1. Floppy einlesen
23969  a410
23970  a410  c9 42             cmp #'b'
23971  a412  f0 2e             beq dbank1       Bank einlesen
23972  a414
23973  a414  c9 55             cmp #'u'
23974  a416  f0 25             beq unit1        fa 1. Floppy einlesen
23975  a418
23976  a418  c9 50             cmp #'p'
23977  a41a  d0 03             bne dos5
23978  a41c
23979  a41c  ===> Untergrenze einlesen <===
23980  a41c
23981  a41c  4c b4 a4          jmp doffl        Adresse Unterrgrenze einlesen
23982  a41f
23983  a41f
23984  a41f  c9 49      dos5   cmp #'i'
23985  a421  f0 75             beq ident        Disk ID einlesen
23986  a423
23987  a423  c9 22             cmp #'"'
23988  a425  f0 07             beq nam01        1. Filenamen einlesen
23989  a427
23990  a427  c9 28             cmp #'('
23991  a429  f0 03             beq nam01        1. Filenamen einlesen
23992  a42b
23993  a42b  4c 6c 79          jmp snerr        Behandlung »?syntax error«
23994  a42e
23995  a42e
23996  a42e  4c dc a4   nam01  jmp name1        1. Filenamen einlesen
23997  a431
23998  a431
23999  a431  ===> »r' überlesen (Ersatzwert) <===
24000  a431
24001  a431  20 80 03  godel1 jsr chrget       nächstes Zeichen nach ac
24002  a434  4c fb a4          jmp delim1       Test/weitere Analyse
24003  a437
24004  a437
24005  a437
```

```
24005  a437  ===> fa 1. Floppy einlesen <===
24006  a437
24007  a437  20 82 a5   on1    jsr on          Abfrage auf folgende Primäradresse,
                                               sonst Fehler
24008  a43a  4c f7 a4   jdel1  jmp del1        ac-Bits in parsts setzen
24009  a43d
24010  a43d
24011  a43d  ===> fa 1. Floppy einlesen <===
24012  a43d
24013  a43d  20 8d a5   unit1  jsr unit        fa Flopy einlesen/prüfen
24014  a440  d0 f8             bne jdel1       immer
24015  a442
24016  a442
24017  a442  ===> Bank einlesen <===
24018  a442
24019  a442  20 9e a5   dbank1 jsr dbank       Bank einlesen
24020  a445  f0 f3             beq jdel1       immer
24021  a447
24022  a447
24023  a447  ===> la einlesen <===
24024  a447
24025  a447  a9 04      logadr lda #_la         la
24026  a449  20 1d a6           jsr prmrpt      snerr, wenn ac-Bits parsts ges.
24027  a44c
24028  a44c  20 f2 a5           jsr getval      la nach xr
24029  a44f  e0 00              cpx #0
24030  a451  f0 42              beq qtyer2      Behandlung »?illegal quantity error'
24031  a453
24032  a453  8e 1b 01           stx dosla       logische Adresse
24033  a456
24034  a456  a9 04              lda #_la
24035  a458  d0 e0              bne jdel1       ac-Bits in parsts setzen
24036  a45a
24037  a45a
24038  a45a  ===> Behandlung »?syntax error« <===
24039  a45a
24040  a45a  4c 6c 79   sner2  jmp snerr        Behandlung »?syntax error«
24041  a45d
24042  a45d
24043  a45d  ===> Record Länge/',s,w' einlesen <===
24044  a45d
24045  a45d  aa         reclen tax             Parameter
24046  a45e
24047  a45e  a9 40             lda #_rcl
24048  a460  20 1d a6          jsr prmrpt      snerr, wenn ac-Bits parsts ges.
24049  a463
24050  a463  e0 57             cpx #'w'
24051  a465  d0 06             bne reco0
24052  a467
24053  a467  ===> WRITE Datei <===
24054  a467
24055  a467  20 80 03          jsr chrget      nächstes Zeichen nach ac
24056  a46a  4c 7b a4          jmp recon       _rcl-Bit setzen, weiter
24057  a46d
24058  a46d
24059  a46d  20 f2 a5   reco0  jsr getval      Record Länge nach xr
24060  a470
24061  a470
```

```
24061  a470  ===> Record Länge prüfen <===
24062  a470
24063  a470  e0 00           cpx #0
24064  a472  f0 21           beq qtyer2      Behandlung »?illegal quantity error«
24065  a474
24066  a474  e0 ff           cpx #255
24067  a476  f0 1d           beq qtyer2      Behandlung »?illegal quantity error«
24068  a478
24069  a478  8e 1e 01        stx dosrcl      Record Länge
24070  a47b
24071  a47b
24072  a47b  ===> _rcl-Bit setzen, weiter <===
24073  a47b
24074  a47b  a9 40      recon lda #_rcl
24075  a47d  d0 14           bne tacky1
24076  a47f
24077  a47f
24078  a47f  ===> 1. Laufwerk einlesen <===
24079  a47f
24080  a47f  a9 10      drv1  lda #_d1        1. Lw
24081  a481  20 1d a6        jsr prmrpt      snerr, wenn ac-Bits parsts ges.
24082  a484
24083  a484  20 f2 a5        jsr getval      1. Lw nach xr
24084  a487  e0 02           cpx #2
24085  a489  b0 0a           bcs qtyer2      Behandlung »?illegal quantity error«
24086  a48b
24087  a48b  8e 12 01        stx dosds1      1. Laufwerk
24088  a48e  8e 14 01        stx dosds2      2. Lw vorbesetzen
24089  a491
24090  a491  a9 10           lda #_d1
24091  a493  d0 62      tacky1 bne del1        ac-Bits in parsts setzen
24092  a495
24093  a495  ===> Behandlung »?illegal quantity error« <===
24094  a495
24095  a495  4c 28 7d   qtyer2 jmp fcerr      Behandlung »?illegal quantity error«
24096  a498
24097  a498
24098  a498  ===> Disk ID einlesen <===
24099  a498
24100  a498  ad 22 01   ident  lda didchk     DOS ID Flag
24101  a49b  d0 bd           bne sner2       ID bereits gelesen
24102  a49d
24103  a49d  20 80 03        jsr chrget      nächstes Zeichen nach ac
24104  a4a0  8d 20 01        sta dosdid      ID erstes Zeichen
24105  a4a3
24106  a4a3  20 80 03        jsr chrget      nächstes Zeichen nach ac
24107  a4a6  8d 21 01        sta dosdid+1    ID zweites Zeichen
24108  a4a9
24109  a4a9  a9 ff           lda #$ff
24110  a4ab  8d 22 01        sta didchk      DOS ID Flag
24111  a4ae
24112  a4ae  20 80 03        jsr chrget      nächstes Zeichen nach ac
24113  a4b1  4c fb a4        jmp delim1      Test/weitere Analyse
24114  a4b4
24115  a4b4
24116  a4b4
```

```
24116  a4b4  ===> Adresse Untergrenze einlesen <===
24117  a4b4
24118  a4b4  a9 02      doffl  lda #_stadr
24119  a4b6  20 22 a6         jsr prxrpt       snerr, wenn ac-Bits parstx ges.
24120  a4b9
24121  a4b9  20 05 a6         jsr getoff       Adresse Untergrenze anch yr/ac
24122  a4bc
24123  a4bc  8c 17 01         sty dosofl       ADDRL Untergrenze BLOAD/BSAVE
24124  a4bf  8d 18 01         sta dosofl+1     ADDRH Untergrenze BLOAD/BSAVE
24125  a4c2
24126  a4c2  a9 02            lda #_stadr
24127  a4c4
24128  a4c4
24129  a4c4  ===> ac-Bits in parstx setzen <===
24130  a4c4
24131  a4c4  05 81      dlimx1 ora parstx       DOS Syntax Byte 2
24132  a4c6  85 81             sta parstx       DOS Syntax Byte 2
24133  a4c8
24134  a4c8  d0 31             bne delim1       Test/weitere Analyse
24135  a4ca
24136  a4ca
24137  a4ca  ===> Adresse Obergrenze einlesen <===
24138  a4ca
24139  a4ca  a9 04      doffh  lda #_endad
24140  a4cc  20 22 a6         jsr prxrpt       snerr, wenn ac-Bits parstx ges.
24141  a4cf
24142  a4cf  20 05 a6         jsr getoff       Adresse Obergrenze anch yr/ac
24143  a4d2
24144  a4d2  8c 19 01         sty dosofh       ADDRL Obergrenze BLOAD/BSAVE
24145  a4d5  8d 1a 01         sta dosofh+1     ADDRH Obergrenze BLOAD/BSAVE
24146  a4d8
24147  a4d8  a9 04            lda #_endad
24148  a4da  d0 e8            bne dlimx1       ac-Bits in parstx setzen
24149  a4dc
24150  a4dc
24151  a4dc  ===> 1. Filenamen einlesen <===
24152  a4dc
24153  a4dc  a9 01      name1  lda #_fn1
24154  a4de  20 b9 a5         jsr newnam       Filenamen einlesen / prüfen
24155  a4e1  8d 11 01         sta dosf1l       Länge Filename 1
24156  a4e4
24157  a4e4  a0 00            ldy #0           Zeiger in Dateinamen
24158  a4e6  20 b7 03   loop6  jsr ind1_1       Zeichen aus fnam nach ac
24159  a4e9  8d 03 ff         sta lcrc         ROM / PGM-RAM einschalten
24160  a4ec  99 b7 12         sta savram,y     und abspeichern
24161  a4ef
24162  a4ef  c8               iny              Zeiger in Dateinamen
24163  a4f0  cc 11 01         cpy dosf1l       Länge Filename 1
24164  a4f3  90 f1            bcc loop6
24165  a4f5
24166  a4f5  a9 01            lda #_fn1
24167  a4f7
24168  a4f7
24169  a4f7  ===> ac-Bits in parsts setzen <===
24170  a4f7
24171  a4f7  05 80      del1   ora parsts       DOS Syntax Byte 1
24172  a4f9  85 80             sta parsts       DOS Syntax Byte 1
24173  a4fb
```

```
24174  a4fb
24175  a4fb  ===> Test/weitere Analyse <===
24176  a4fb
24177  a4fb  20 86 03   delim1 jsr chrgot       letztes Zeichen erneut nach ac
24178  a4fe  d0 19             bne nxxx         Kommatest, nächster Parameter
24179  a500
24180  a500  4c ea a3   done1  jmp done         Analyse beendet
24181  a503
24182  a503
24183  a503  ===> Test auf folg. ON (Disk-BASIC) <===
24184  a503
24185  a503  c9 91      next6  cmp #_ont        ON Token
24186  a505  d0 03             bne next6a       Test auf folgendes TO
24187  a507
24188  a507  ===> fa 1. Floppy einlesen <===
24189  a507
24190  a507  4c 37 a4          jmp on1          fa 1. Floppy einlesen
24191  a50a
24192  a50a
24193  a50a  ===> Test auf folgendes TO <===
24194  a50a
24195  a50a  c9 a4      next6a cmp #_tot        TO Token
24196  a50c  f0 02             beq next6b       Test auf folgendes »p«
24197  a50e  d0 6d             bne sner3        Behandlung »?syntax error«
24198  a510
24199  a510
24200  a510  ===> Test auf folgendes »p« <===
24201  a510
24202  a510  20 80 03   next6b jsr chrget       nächstes Zeichen nach ac
24203  a513  c9 50             cmp #'p'
24204  a515  d0 0f             bne pars22
24205  a517  f0 b1             beq doffh        Adresse Obergrenze einlesen
24206  a519
24207  a519
24208  a519  ===> Kommatest, nächster Parameter <===
24209  a519
24210  a519  c9 2c      nxxx   cmp #','
24211  a51b  d0 e6             bne next6        Test auf folg. ON (Disk-BASIC)
24212  a51d
24213  a51d  20 80 03          jsr chrget       nächstes Zeichen nach ac
24214  a520  4c f8 a3          jmp parse1       Disk-Parameter 1 einlesen
24215  a523
24216  a523
24217  a523  ===> Disk-Parameter 2 einlesen <===
24218  a523
24219  a523  20 80 03   parse2 jsr chrget       nächstes Zeichen nach ac
24220  a526  c9 44      pars22 cmp #'d'
24221  a528  f0 10             beq drv2         2. Laufwerk einlesen
24222  a52a
24223  a52a  c9 91             cmp #_ont        ON Token
24224  a52c  f0 1f             beq on2          fa 2. Flopy einlesen
24225  a52e
24226  a52e  c9 55             cmp #'u'
24227  a530  f0 21             beq unit2        fa 2. Floppy einlesen
24228  a532
24229  a532  c9 22             cmp #'"'
24230  a534  f0 22             beq name2        Filenamen 2 einlesen (Disk-BASIC)
24231  a536
```

```
24232  a536  c9 28            cmp #'('
24233  a538  f0 1e            beq name2        Filenamen 2 einlesen (Disk-BASIC)
24234  a53a
24235  a53a
24236  a53a  ===> 2. Laufwerk einlesen <===
24237  a53a
24238  a53a  a9 20      drv2  lda #_d2
24239  a53c  20 1d a6         jsr prmrpt       snerr, wenn ac-Bits parsts ges.
24240  a53f
24241  a53f  20 f2 a5         jsr getval       2. Laufwerk nach xr
24242  a542  e0 02            cpx #2
24243  a544  b0 39            bcs qtyerr       Behandlung »?illegal quantity error«
24244  a546
24245  a546  8e 14 01         stx dosds2       2. Laufwerk
24246  a549
24247  a549  a9 20            lda #_d2
24248  a54b  d0 1b            bne del2         ac-Bits in parsts setzen, weiter
24249  a54d
24250  a54d
24251  a54d  ===> fa 2. Flopy einlesen <===
24252  a54d
24253  a54d  20 82 a5   on2   jsr on           Abfrage auf folgende Primäradresse,
                                               sonst Fehler
24254  a550  4c 68 a5         jmp del2         ac-Bits in parsts setzen, weiter
24255  a553
24256  a553
24257  a553  ===> fa 2. Floppy einlesen <===
24258  a553
24259  a553  20 8d a5   unit2  jsr unit         fa Flopy einlesen/prüfen
24260  a556  d0 10            bne del2         ac-Bits in parsts setzen, weiter
24261  a558
24262  a558
24263  a558  ===> Filenamen 2 einlesen (Disk-BASIC) <===
24264  a558
24265  a558  a9 02     name2  lda #_fn2
24266  a55a  20 b9 a5         jsr newnam       Filenamen einlesen / prüfen
24267  a55d
24268  a55d  8d 13 01         sta dosf2l       Länge 2. Filename
24269  a560  8e 15 01         stx dosf2a       ADDRL 2. Filename
24270  a563  8c 16 01         sty dosf2a+1     ADDRH 2. Filename
24271  a566
24272  a566  a9 02            lda #_fn2
24273  a568
24274  a568
24275  a568  ===> ac-Bits in parsts setzen, weiter <===
24276  a568
24277  a568  05 80      del2  ora parsts       DOS Syntax Byte 1
24278  a56a  85 80            sta parsts       DOS Syntax Byte 1
24279  a56c
24280  a56c  20 86 03         jsr chrgot       letztes Zeichen erneut nach ac
24281  a56f  f0 8f            beq done1        Analyse beendet
24282  a571
24283  a571  c9 2c            cmp #','
24284  a573  f0 ae            beq parse2       Disk-Parameter 2 einlesen
24285  a575
24286  a575  c9 91            cmp #_ont        ON Token
24287  a577  f0 d4            beq on2          fa 2. Flopy einlesen
24288  a579
```

```
24289  a579  c9 55            cmp #'u'
24290  a57b  f0 d6            beq unit2        fa 2. Floppy einlesen
24291  a57d
24292  a57d  ===> Behandlung »?illegal quantity error« <===
24293  a57d
24294  a57d
24295  a57d  ===> Behandlung »?syntax error« <===
24296  a57d
24297  a57d  d0 37      sner3  bne sner         Behandlung »?syntax error«
24298  a57f
24299  a57f  4c 28 7d   qtyerr jmp fcerr        Behandlung »?illegal quantity error«
24300  a582
24301  a582
24302  a582  ===> Abfrage auf folgende Primäradresse, sonst Fehler <===
24303  a582
24304  a582  20 80 03   on     jsr chrget       nächstes Zeichen nach ac
24305  a585  c9 42            cmp #'b'
24306  a587  f0 15            beq dbank        Bank einlesen
24307  a589
24308  a589  c9 55            cmp #'u'
24309  a58b  d0 29            bne sner         Behandlung »?syntax error«
24310  a58d
24311  a58d
24312  a58d  ===> fa Flopy einlesen/prüfen <===
24313  a58d
24314  a58d  20 f2 a5   unit   jsr getval       Geräteadresse Floppy
24315  a590
24316  a590  ===> Zulässigkeit Geräteadresse prüfen <===
24317  a590
24318  a590  e0 1f            cpx #31
24319  a592  b0 56            bcs errild       Behandlung »?illegal device error«
24320  a594
24321  a594  e0 04            cpx #4
24322  a596  90 52            bcc errild       Behandlung »?illegal device error«
24323  a598
24324  a598  8e 1c 01         stx dosfa        Primäradresse
24325  a59b
24326  a59b  a9 08            lda #_fa
24327  a59d  60               rts
24328  a59e
24329  a59e
24330  a59e  ===> Bank einlesen <===
24331  a59e
24332  a59e  a9 01      dbank lda #_bnk
24333  a5a0  20 22 a6         jsr prxrpt       snerr, wenn ac-Bits parstx ges.
24334  a5a3
24335  a5a3  20 f2 a5         jsr getval       Bank nach xr
24336  a5a6  e0 10            cpx #16
24337  a5a8  b0 d5            bcs qtyerr       Behandlung »?illegal quantity error«
24338  a5aa
24339  a5aa  8e 1f 01         stx dosbnk       Bank Nummer
24340  a5ad
24341  a5ad  a9 01            lda #_bnk
24342  a5af  05 81            ora parstx       DOS Syntax Byte 2
24343  a5b1  85 81            sta parstx       DOS Syntax Byte 2
24344  a5b3
24345  a5b3  a9 00            lda #0           Flags setzen
24346  a5b5  60               rts
```

```
24347  a5b6
24348  a5b6
24349  a5b6
24349  a5b6  ===> Behandlung »?syntax error« <===
24350  a5b6
24351  a5b6  4c 6c 79    sner    jmp snerr         Behandlung »?syntax error«
24352  a5b9
24353  a5b9
24354  a5b9  ===> Filenamen einlesen / prüfen <===
24355  a5b9
24356  a5b9  20 1d a6    newnam  jsr prmrpt        snerr, wenn ac-Bits parsts ges.
24357  a5bc
24358  a5bc  20 7b 87            jsr sav13        Filename in Arbeitsstring, frestr
24359  a5bf  aa                  tax              Länge Filename
24360  a5c0  f0 25               beq errmfn       Behandlung »?missing filename«
24361  a5c2
24362  a5c2  a0 00               ldy #0           Zeiger auf 1. Zeichen Filename
24363  a5c4  20 b7 03            jsr ind1_1       1. Zeichen nach ac
24364  a5c7  c9 40               cmp #'@'         überschreiben
24365  a5c9  d0 12               bne lenchk       Länge des Filenamen prüfen
24366  a5cb
24367  a5cb  a9 80               lda #_rpl
24368  a5cd  20 1d a6            jsr prmrpt       snerr, wenn ac-Bits parsts ges.
24369  a5d0
24370  a5d0  a5 80               lda parsts       DOS Syntax Byte 1
24371  a5d2  09 80               ora #_rpl
24372  a5d4  85 80               sta parsts       DOS Syntax Byte 1
24373  a5d6
24374  a5d6  ca                  dex              Stringlänge - @
24375  a5d7  e6 24               inc index1       ADDRL Filename
24376  a5d9  d0 02               bne lenchk       Länge des Filenamen prüfen
24377  a5db  e6 25               inc index1+1     ADDRH Filename
24378  a5dd
24379  a5dd
24380  a5dd  ===> Länge des Filenamen prüfen <===
24381  a5dd
24382  a5dd  8a          lenchk  txa              fnlen
24383  a5de  c9 11               cmp #17
24384  a5e0  b0 0b               bcs errlen       Behandlung »?string too long«
24385  a5e2
24386  a5e2  a6 24               ldx index1
24387  a5e4  a4 25               ldy index1+1
24388  a5e6  60                  rts
24389  a5e7
24390  a5e7
24391  a5e7  ===> Behandlung »?missing filename« <===
24392  a5e7
24393  a5e7  a2 08       errmfn  ldx #_ermfn
24394  a5e9  2c                  .byte _skip2
24395  a5ea
24396  a5ea
24397  a5ea  ===> Behandlung »?illegal device error« <===
24398  a5ea
24399  a5ea  a2 09       errild  ldx #_eridn
24400  a5ec  2c                  .byte _skip2
24401  a5ed
24402  a5ed
```

```
24403  a5ed  ===> Behandlung »?string too long« <===
24404  a5ed
24405  a5ed  a2 17      errlen ldx #_erls
24406  a5ef  4c 3c 4d          jmp error        ind. Behandlung Fehlermeldung
24407  a5f2
24408  a5f2
24409  a5f2  ===> 8-Bit INT (eingekl.) nach xr <===
24410  a5f2
24411  a5f2  20 80 03   getval jsr chrget       nächstes Zeichen nach ac
24412  a5f5  f0 bf             beq sner         Behandlung »?syntax error«
24413  a5f7  90 09             bcc gtv15        Ziffer
24414  a5f9
24415  a5f9  20 59 79          jsr chkopn       Test auf folg. »(«, sonst Fehler
24416  a5fc  20 f4 87          jsr getbyt       8 Bit INT üb. frmnum nach xr
24417  a5ff  4c 56 79          jmp chkcls       Test auf folg. »)«, sonst Fehler
24418  a602
24419  a602
24420  a602  4c f4 87   gtv15  jmp getbyt       8 Bit INT üb. frmnum nach xr
24421  a605
24422  a605
24423  a605  ===> 16-Bit INT (eingekl.) nach yr/ac <===
24424  a605
24425  a605  20 80 03   getoff jsr chrget       nächstes Zeichen nach ac
24426  a608  f0 ac             beq sner         Behandlung »?syntax error«
24427  a60a  90 0e             bcc gtff5        Ziffer
24428  a60c
24429  a60c  20 59 79          jsr chkopn       Test auf folg. »(«, sonst Fehler
24430  a60f  20 12 88          jsr getwrd       16 Bit INT nach linnum
24431  a612  20 56 79          jsr chkcls       Test auf folg. »)«, sonst Fehler
24432  a615
24433  a615  a4 16             ldy linnum
24434  a617  a5 17             lda linnum+1
24435  a619  60                rts
24436  a61a
24437  a61a
24438  a61a  4c 12 88   gtff5  jmp getwrd       16 Bit INT nach linnum
24439  a61d
24440  a61d
24441  a61d  ===> snerr, wenn ac-Bits parsts ges. <===
24442  a61d
24443  a61d  25 80      prmrpt and parsts       DOS Syntax Byte 1
24444  a61f  d0 95             bne sner         Behandlung »?syntax error«
24445  a621
24446  a621  60                rts
24447  a622
24448  a622
24449  a622  ===> snerr, wenn ac-Bits parstx ges. <===
24450  a622
24451  a622  25 81      prxrpt and parstx       DOS Syntax Byte 2
24452  a624  d0 90             bne sner         Behandlung »?syntax error«
24453  a626
24454  a626  60                rts
24455  a627
24456  a627
24457  a627  ===> Tab. der Parameter für Befehlsstrings <===
24458  a627
24459  a627            tabld
24460  a627
```

```
24461  a627
24462  a627  ===> DCLEAR (i0) <===
24463  a627
24464  a627  49 d1      pdclr  .byte 'i', _drv1
24465  a629
24466  a629
24466  a629  ===> CATALOG, DIRECTORY ($0:fnam1) <===
24467  a629
24468  a629  24 d1 3a   pdir   .byte '$', _drv1, ':', _fnam1
24468  a62c  f1
24469  a62d
24470  a62d  ===> DOPEN, DLOAD, BLOAD, DSAVE, BSAVE, DVERIFY <===
24471  a62d
24472  a62d  f0 d1 3a   pdop   .byte _ovwr, _drv1, ':', _fnam1, ',', _seqwr, ',',
                                    _recln
24472  a630  f1 2c e1
24472  a633  2c e0
24473  a635
24474  a635  ===> CONCAT (c0:fnam2=0:fnam2,1:fnam1) <===
24475  a635
24476  a635  43 d2 3a   pconc  .byte 'c', _drv2, ':', _fnam2, '='
24476  a638  f2 3d
24477  a63a  d2 3a f2          .byte _drv2, ':', _fnam2, ','
24477  a63d  2c
24478  a63e
24479  a63e  ===> APPEND (0:fnam,a) <===
24480  a63e
24481  a63e  d1 3a f1   papp   .byte _drv1, ':', _fnam1, ',a'
24481  a641  2c 41
24482  a643
24483  a643  ===> HEADER (n0:header,id) <===
24484  a643
24485  a643  4e d1 3a   phed   .byte 'n', _drv1, ':', _fnam1, ',', _id
24485  a646  f1 2c d0
24486  a649
24487  a649  ===> COLLECT <===
24488  a649
24489  a649  56 d1      pcoll  .byte 'v', _drv1
24490  a64b
24491  a64b  ===> BACKUP (d1=0) <===
24492  a64b
24493  a64b  44 d2 3d   pbak   .byte 'd', _drv2, '=', _drv1
24493  a64e  d1
24494  a64f
24495  a64f  ===> DCOPY (c1:fnam2=0:fnam1) <===
24496  a64f
24497  a64f  43 d2 3a   pdcop  .byte 'c', _drv2, ':', _fnam2, '=', _drv1, ':',
                                    _fnam1
24497  a652  f2 3d d1
24497  a655  3a f1
24498  a657
24499  a657  ===> RENAME (r0:fnam2=0:fnam1) <===
24500  a657
24501  a657  52 d1 3a   pren   .byte 'r', _drv1, ':', _fnam2, '=', _drv1, ':',
                                    _fnam1
24501  a65a  f2 3d d1
24501  a65d  3a f1
24502  a65f
```

```
24503  a65f  ===> SCRATCH (s0:fnam1) <===
24504  a65f
24505  a65f  53 d1 3a    pscr   .byte 's', _drv1, ':', _fnam1
24505  a662  f1
24506  a663
24507  a663
24507  a663  ===> RECORD (p sa rlrh bp)
24508  a663
24509  a663  50 c2 e2    prec   .byte 'p', _sa, _rlrh, _recln
24509  a666  e0
24510  a667
24511  a667
24512  a667  ===> DOS-Befehlsstring zusammensetzen u. ausgeben <===
24513  a667
24514  a667  8d 10 01    sendp  sta xcnt          DOS Schleifen-Zähler
24515  a66a
24516  a66a  98                 tya              Zeiger in tabld
24517  a66b  48                 pha              Zeiger in tabld
24518  a66c
24519  a66c  20 0d a8           jsr oldclr       DS$-String ungültig machen
24520  a66f
24521  a66f  a2 00              ldx #0           Zeiger in Befehlsstring
24522  a671
24523  a671
24524  a671  ===> nächster Term Befehlsstring <===
24525  a671
24526  a671  68          sdp1   pla              Zeiger in tabld
24527  a672
24528  a672  ce 10 01           dec xcnt         DOS Schleifen-Zähler
24529  a675  30 48             bmi tranr        Adr. Befehlsstr., la, fa, sa nach ZP
24530  a677
24531  a677  a8                 tay
24532  a678  c8                 iny              Zeiger in tabld
24533  a679
24534  a679  98                 tya
24535  a67a  48                 pha              Zeiger in tabld
24536  a67b
24537  a67b  b9 27 a6           lda tabld,y      Zeichen aus Tabelle
24538  a67e  10 37              bpl sdp5         ASCII nach Befehlsstring
24539  a680
24540  a680  c9 c2              cmp #_sa         Sekundär Adresse
24541  a682  f0 52              beq rsca         sa in Befehlsstring eintragen
24542  a684
24543  a684  c9 d0              cmp #_id         ID
24544  a686  f0 5d              beq rid          Disk ID in Befehlsstring
24545  a688
24546  a688  c9 e2              cmp #_rlrh       Recordnummer (low/high)
24547  a68a  f0 77              beq rdcn         Recordnummer low/high in Befehlsstr.
24548  a68c
24549  a68c  c9 e1              cmp #_seqwr      »,s,w«
24550  a68e  f0 61              beq rwrt         dosrcl in Befehlsstring
24551  a690
24552  a690  c9 f0              cmp #_ovwr       »@«
24553  a692  f0 47              beq rfat         Test _rpl/ @ in Befehlsstr.
24554  a694
24555  a694  c9 f1              cmp #_fnam1      1. Filename
24556  a696  f0 75              beq rsfn         1. Filename in Befehlsstring
24557  a698
```

```
24558  a698  c9 f2           cmp #_fnam2      2. Filename
24559  a69a  f0 21           beq gordfn       Zeiger auf 2. Filenamen
24560  a69c
24561  a69c  c9 e0           cmp #_recln      Record Länge
24562  a69e  d0 05           bne sdp2
24563  a6a0
24564  a6a0
24564  a6a0  ===> dosrcl nach Befehlsstring <===
24565  a6a0
24566  a6a0  ad 1e 01        lda dosrcl       Record Länge
24567  a6a3  d0 12           bne sdp5         ac, xr in DOS-Befehlsstring
24568  a6a5
24569  a6a5
24570  a6a5  c9 d1     sdp2  cmp #_drv1       1. Laufwerk
24571  a6a7  d0 05           bne sdp3
24572  a6a9
24573  a6a9  ===> 1. Laufwerk nach Befehlsstring <===
24574  a6a9
24575  a6a9  ad 12 01        lda dosds1       1. Laufwerk
24576  a6ac  10 07           bpl sdp4         ac nach ASCII in Befehlsstr.
24577  a6ae
24578  a6ae
24579  a6ae  c9 d2     sdp3  cmp #_drv2       2. Laufwerk
24580  a6b0  d0 bf           bne sdp1         nächster Term Befehlsstring
24581  a6b2
24582  a6b2  ===> 2. Laufwerk nach Befehlsstring <===
24583  a6b2
24584  a6b2  ad 14 01        lda dosds2       2. Laufwerk
24585  a6b5
24586  a6b5
24587  a6b5  ===> ac nach ASCII in Befehlsstr. <===
24588  a6b5
24589  a6b5  09 30     sdp4  ora #'0'
24590  a6b7
24591  a6b7
24592  a6b7  ===> ac, xr in DOS-Befehlsstring <===
24593  a6b7
24594  a6b7  9d 00 11  sdp5  sta dosstr,x     DOS Kommando String Puffer
24595  a6ba
24596  a6ba  e8              inx              Zeiger in Befehlsstring
24597  a6bb  d0 b4           bne sdp1         nächster Term Befehlsstring
24598  a6bd
24599  a6bd
24600  a6bd  ===> Zeiger auf 2. Filenamen <===
24601  a6bd
24602  a6bd  f0 64     gordfn beq rdfn        2. Filename in Befehlsstring
24603  a6bf
24604  a6bf
24605  a6bf  ===> Adr. Befehlsstr., la, fa, sa nach ZP <===
24606  a6bf
24607  a6bf  8a        tranr  txa             Länge Befehlsstring
24608  a6c0  48               pha             Länge Befehlsstring
24609  a6c1
24610  a6c1  a2 00            ldx #<dosstr    DOS Kommando String Puffer
24611  a6c3  a0 11            ldy #>dosstr    DOS Kommando String Puffer
24612  a6c5  20 5d 92         jsr csetnm      ac, xr/yr nach fnlen, fnadr/+1
24613  a6c8
24614  a6c8  ad 1b 01         lda dosla       logische Adresse
```

```
24615  a6cb  ae 1c 01          ldx dosfa        Primäradresse
24616  a6ce  ac 1d 01          ldy dossa        Sekundäradresse
24617  a6d1  20 57 92          jsr cstlfs       ac, xr, yr nach la, fa, sa
24618  a6d4
24619  a6d4  68                pla              Länge Befehlsstring
24620  a6d5  60                rts
24621  a6d6
24622  a6d6
24623  a6d6  ===> sa in Befehlsstring eintragen <===
24624  a6d6
24625  a6d6  ad ed 11    rsca  lda tdossa
24626  a6d9  d0 dc             bne sdp5         ac, xr in DOS-Befehlsstring
24627  a6db
24628  a6db
24629  a6db  ===> Test _rpl/ @ in Befehlsstr. <===
24630  a6db
24631  a6db  24 80       rfat  bit parsts       DOS Syntax Byte 1
24632  a6dd
24633  a6dd   7.......  1 = überschreiben (_rpl)
24634  a6dd
24635  a6dd  30 02             bmi rfata        @ in Befehlsstring
24636  a6df  10 90             bpl sdp1         nächster Term Befehlsstring
24637  a6e1
24638  a6e1
24639  a6e1  ===> @ in Befehlsstring <===
24640  a6e1
24641  a6e1  a9 40       rfata lda #'@'
24642  a6e3  d0 d2             bne sdp5         ac, xr in DOS-Befehlsstring
24643  a6e5
24644  a6e5
24645  a6e5  ===> Disk ID in Befehlsstring <===
24646  a6e5
24647  a6e5  ad 20 01    rid   lda dosdid       ID erstes Zeichen
24648  a6e8  9d 00 11          sta dosstr,x     DOS Kommando String Puffer
24649  a6eb
24650  a6eb  e8                inx              Zeiger in Befehlsstring
24651  a6ec  ad 21 01          lda dosdid+1     ID zweites Zeichen
24652  a6ef  d0 c6             bne sdp5         ac, xr in DOS-Befehlsstring
24653  a6f1
24654  a6f1
24655  a6f1  ===> dosrcl in Befehlsstring <===
24656  a6f1
24657  a6f1  ad 1e 01    rwrt  lda dosrcl       Record Länge
24658  a6f4  f0 04             beq rwrt1        SEQ File
24659  a6f6
24660  a6f6  ===> »l« für Record Länge in Befehlsstring <===
24661  a6f6
24662  a6f6  a9 4c             lda #'l'
24663  a6f8  d0 bd             bne sdp5         ac, xr in DOS-Befehlsstring
24664  a6fa
24665  a6fa
24666  a6fa  ===> »,s,w« in Befehlsstring <===
24667  a6fa
24668  a6fa  a9 53       rwrt1 lda #'s'
24669  a6fc  8d 1e 01          sta dosrcl       Record Länge
24670  a6ff
24671  a6ff  a9 57             lda #'w'
24672  a701  d0 b4             bne sdp5         ac, xr in DOS-Befehlsstring
```

```
24673  a703
24674  a703
24675  a703  ===> Recordnummer low/high in Befehlsstr. <===
24676  a703
24677  a703  a5 16      rdcn   lda linnum       Record Nummer low
24678  a705  9d 00 11          sta dosstr,x     DOS Kommando String Puffer
24679  a708
24680  a708  a5 17             lda linnum+1     Record Nummer high
24681  a70a  e8               inx              Zeiger in Befehlsstring
24682  a70b  d0 aa            bne sdp5         ac, xr in DOS-Befehlsstring
24683  a70d
24684  a70d
24685  a70d  ===> 1. Filename in Befehlsstring <===
24686  a70d
24687  a70d  ac 11 01   rsfn   ldy dosf1l       Länge Filename 1
24688  a710  f0 33            beq rdrt0
24689  a712
24690  a712  a0 00            ldy #0           Zeiger in Filenamen
24691  a714
24692  a714  b9 b7 12   rsmov  lda savram,y     Filenamen nach
24693  a717  9d 00 11          sta dosstr,x     Befehlsstring
24694  a71a
24695  a71a  e8               inx              Zeiger in Befehlsstring
24696  a71b  c8               iny              Zeiger in Filenamen
24697  a71c
24698  a71c  cc 11 01          cpy dosf1l       Länge Filename 1
24699  a71f  d0 f3            bne rsmov
24700  a721  f0 23            beq rdrt1
24701  a723
24702  a723
24703  a723  ===> 2. Filename in Befehlsstring <===
24704  a723
24705  a723  ad 15 01   rdfn   lda dosf2a       ADDRL 2. Filename
24706  a726  85 24            sta index1       ADDRL 2. Filename
24707  a728
24708  a728  ad 16 01          lda dosf2a+1     ADDRH 2. Filename
24709  a72b  85 25            sta index1+1     ADDRH 2. Filename
24710  a72d
24711  a72d  ac 13 01          ldy dosf2l       Länge 2. Filename
24712  a730  f0 13            beq rdrt0
24713  a732
24714  a732  a0 00            ldy #0           Zeiger in Filenamen
24715  a734
24716  a734  20 b7 03   rdmov  jsr ind1_1       Filenamen
24717  a737  8d 03 ff          sta lcrc         nach
24718  a73a  9d 00 11          sta dosstr,x     Befehlsstring
24719  a73d
24720  a73d  e8               inx              Zeiger in Befehlsstring
24721  a73e  c8               iny              Zeiger in Filenamen
24722  a73f
24723  a73f  cc 13 01          cpy dosf2l       Länge 2. Filename
24724  a742  d0 f0            bne rdmov
24725  a744  24               .byte _skip1
24726  a745
24727  a745  ca         rdrt0  dex
24728  a746  4c 71 a6    rdrt1  jmp sdp1         nächster Term Befehlsstring
24729  a749
24730  a749
```

```
24731  a749  ===> Syntaxprüfung HEADER, SCRATCH <===
24732  a749
24733  a749  29 e6      chk1   and #_rpl+_rcl+_d2+_la+_fn2
24734  a74b  f0 03             beq chk2          Syntaxprüfung DLOAD, DSAVE
24735  a74d
24736  a74d  ===> Behandlung »?syntax error« <===
24737  a74d
24738  a74d  4c 6c 79   chker1 jmp snerr         Behandlung »?syntax error«
24739  a750
24740  a750
24741  a750  ===> Syntaxprüfung DLOAD, DSAVE <===
24742  a750
24743  a750  a5 80      chk2   lda parsts        DOS Syntax Byte 1
24744  a752  29 01             and #_fn1
24745  a754  c9 01             cmp #_fn1
24746  a756  d0 f5             bne chker1        Behandlung »?syntax error«
24747  a758
24748  a758  a5 80             lda parsts        DOS Syntax Byte 1
24749  a75a  60                rts
24750  a75b
24751  a75b
24752  a75b  ===> Syntaxprüfung CATALOG, COLLECT <===
24753  a75b
24754  a75b  29 e7      chk3   and #_rpl+_rcl+_d2+_la+_fn2+_fn1
24755  a75d  d0 ee             bne chker1        Behandlung »?syntax error«
24756  a75f
24757  a75f  60                rts
24758  a760
24759  a760
24760  a760  ===> Syntaxprüfung CONCAT, COPY <===
24761  a760
24762  a760  29 c4      chk4   and #_rpl+_rcl+_la
24763  a762  d0 e9             bne chker1        Behandlung »?syntax error«
24764  a764
24765  a764  a5 80             lda parsts        DOS Syntax Byte 1
24766  a766
24767  a766
24768  a766  ===> Syntaxprüfung RENAME <===
24769  a766
24770  a766  29 03      chk5   and #_fn2+_fn1
24771  a768  c9 03             cmp #_fn2+_fn1
24772  a76a  d0 e1             bne chker1        Behandlung »?syntax error«
24773  a76c
24774  a76c  a5 80             lda parsts        DOS Syntax Byte 1
24775  a76e  60                rts
24776  a76f
24777  a76f
24778  a76f  ===> Syntaxprüfung APPEND, DOPEN <===
24779  a76f
24780  a76f  29 05      chk6   and #_la+_fn1
24781  a771  c9 05             cmp #_la+_fn1
24782  a773  d0 d8             bne chker1        Behandlung »?syntax error«
24783  a775
24784  a775  a5 80             lda parsts        DOS Syntax Byte 1
24785  a777  60                rts
24786  a778
24787  a778
```

```
24788  a778  ===> DS$ über Sek.adr. 15 einlesen <===
24789  a778
24790  a778  a5 7a      errchl lda dsdesc       Desc. DS$ (Länge)
24791  a77a  d0 19             bne echks
24792  a77c
24793  a77c  ===> DS$-String anlegen <===
24794  a77c
24795  a77c  a9 28             lda #_dsdln     Länge DS$-String
24796  a77e  85 7a             sta dsdesc      Desc. DS$ (Länge)
24797  a780
24798  a780  20 99 92          jsr getspa      Arbeitsstring Länge ac anlegen
24799  a783  86 7b             stx dsdesc+1    Desc. DS$ (ADDRL)
24800  a785  84 7c             sty dsdesc+2    Desc. DS$ (ADDRH)
24801  a787
24802  a787  ===> Trailer hinter DS$-STring <===
24803  a787
24804  a787  a0 28             ldy #_dsdln     Zeiger auf DS$-Trailer
24805  a789  8d 04 ff          sta lcrd        ROM / VAR-RAM einschalten
24806  a78c
24807  a78c  a9 7a             lda #<dsdesc    Desc. DS$ (Länge)
24808  a78e  91 7b             sta (dsdesc+1),y Desc. DS$ (ADDRL)
24809  a790
24810  a790  c8                iny
24811  a791  a9 00             lda #>dsdesc    Desc. DS$ (Länge)
24812  a793  91 7b             sta (dsdesc+1),y Desc. DS$ (ADDRL)
24813  a795
24814  a795  ae 1c 01   echks  ldx dosfa       fa
24815  a798  d0 05             bne eread
24816  a79a
24817  a79a  ===> Vorgabe fa Floppy eintragen <===
24818  a79a
24819  a79a  a2 08             ldx #_deffa
24820  a79c  8e 1c 01          stx dosfa       Primäradresse
24821  a79f
24822  a79f  a9 00      eread  lda #_defla
24823  a7a1  a0 6f             ldy #_talk+_cmdsa      sa
24824  a7a3  20 57 92          jsr cstlfs      ac, xr, yr nach la, fa, sa
24825  a7a6
24826  a7a6  a9 00             lda #0          fnlen
24827  a7a8  20 5d 92          jsr csetnm      ac, xr/yr nach fnlen, fnadr/+1
24828  a7ab  20 d8 90          jsr copen       BASIC-Statement OPEN ($9f)
24829  a7ae
24830  a7ae  a2 00             ldx #_defla
24831  a7b0  20 c6 ff          jsr kchkin      Eingabekanal üb. Vector öffnen
                                               (Sprungv.)
24832  a7b3  b0 20             bcs errbad      Behandl. Fehler Einlesen DS$-String
24833  a7b5
24834  a7b5  a0 ff             ldy #$ff        Zeiger in DS$-String
24835  a7b7  c8         loop1  iny
24836  a7b8  20 63 92          jsr cbasin      ind. Zeich. akt. Kanal nach ac
24837  a7bb  8d 04 ff          sta lcrd        ROM / VAR-RAM einschalten
24838  a7be
24839  a7be  c9 0d             cmp #_cr
24840  a7c0  f0 06             beq errend      Abschluß Einlesen DS$-String
24841  a7c2
24842  a7c2  91 7b             sta (dsdesc+1),y in DS$-String
24843  a7c4
24844  a7c4  c0 28             cpy #_dsdln
```

```
24845  a7c6  90 ef            bcc loop1
24846  a7c8
24847  a7c8
24848  a7c8  ===> Abschluß Einlesen DS$-String <===
24849  a7c8
24850  a7c8  a9 00      errend lda #_eom
24851  a7ca  91 7b            sta (dsdesc+1),y in DS$-String
24852  a7cc
24853  a7cc  20 6f 92         jsr cclrch       ind. I/O-Kanäle schließen
24854  a7cf  a9 00            lda #_defla      la
24855  a7d1  38               sec
24856  a7d2  4c 75 92         jmp cclose       log. File (la in ac) schließen
24857  a7d5
24858  a7d5
24859  a7d5  ===> Behandl. Fehler Einlesen DS$-String <===
24860  a7d5
24861  a7d5  48         errbad pha             Fehlernummer
24862  a7d6
24863  a7d6  20 c8 a7         jsr errend       Abschluß Einlesen DS$-String
24864  a7d9  20 0d a8         jsr oldclr       DS$-String ungültig machen
24865  a7dc
24866  a7dc  68               pla
24867  a7dd  aa               tax              Fehlernummer
24868  a7de  4c 3c 4d         jmp error        ind. Behandlung Fehlermeldung
24869  a7e1
24870  a7e1
24871  a7e1  ===> Bearbeitung »are you sure ?« <===
24872  a7e1
24873  a7e1  24 7f      rusure bit runmod       Modus: $80=PGM, $40=OVL, $00=DIR
24874  a7e3  30 25            bmi ans3         PGM Modus
24875  a7e5
24876  a7e5  20 81 92         jsr cprimm       folg. Text ausg./überspringen
24877  a7e8
24878  a7e8  41 52 45         .byte »are you sure?«, _eom
24878  a7eb  20 59 4f
24878  a7ee  55 20 53
24878  a7f1  55 52 45
24878  a7f4  3f 00
24879  a7f6
24880  a7f6  20 6f 92         jsr cclrch       ind. I/O-Kanäle schließen
24881  a7f9  20 63 92         jsr cbasin       ind. Zeich. akt. Kanal nach ac
24882  a7fc  48               pha              erstes Zeichen
24883  a7fd
24884  a7fd  c9 0d      ans1  cmp #_cr
24885  a7ff  f0 05            beq ans2
24886  a801
24887  a801  20 63 92         jsr cbasin       ind. Zeich. akt. Kanal nach ac
24888  a804  d0 f7            bne ans1
24889  a806
24890  a806  68         ans2  pla              erstes Zeichen
24891  a807  c9 59            cmp #'y'
24892  a809  60               rts
24893  a80a
24894  a80a
24895  a80a  a9 00      ans3  lda #0           EQUAL FLAG = 1 (y)
24896  a80c  60               rts
24897  a80d
24898  a80d
```

```
24899  a80d  ===> DS$-String ungültig machen <===
24900  a80d
24901  a80d  98          oldclr tya
24902  a80e  48                 pha              yr
24903  a80f
24904  a80f  a5 7a              lda dsdesc       Desc. DS$ (Länge)
24905  a811  f0 0d              beq oldcl1       schon ungültig
24906  a813
24907  a813  ===> Trailer DS$-String ungültig machen <===
24908  a813
24909  a813  a0 28              ldy #_dsdln
24910  a815  98                 tya              Länge DS$-String
24911  a816  8d 04 ff           sta lcrd         ROM / VAR-RAM einschalten
24912  a819
24913  a819  91 7b              sta (dsdesc+1),y Trailer: Länge ungültiger String
24914  a81b
24915  a81b  c8                 iny
24916  a81c  a9 ff              lda #$ff
24917  a81e  91 7b              sta (dsdesc+1),y Trailer: Flag String ungültig
24918  a820
24919  a820  a9 00       oldcl1 lda #0
24920  a822  8d 03 ff           sta lcrc         ROM / PGM-RAM einschalten
24921  a825  85 7a              sta dsdesc       Desc. DS$ (Länge)
24922  a827
24923  a827  68                 pla              yr
24924  a828  a8                 tay
24925  a829
24926  a829  60                 rts
24927  a82a
24928  a82a
24929  a82a  ===> Flickwerk <===
24930  a82a
24931  a82a
24932  a82a  ===> Text 'key 0,' <===
24933  a82a
24934  a82a  2c 30 20    preamb .byte ',0 yek'
24934  a82d  59 45 4b
24935  a830
24936  a830
24937  a830  ===> 8-Bit INT nach ASCII auf akt. Kanal <===
24938  a830
24939  a830  aa          prtdec tax              8-Bit INT
24940  a831
24941  a831  98                 tya
24942  a832  48                 pha              yr
24943  a833
24944  a833  a9 00              lda #0           High-Byte
24945  a835  20 32 8e           jsr linprt       xr/ac von hex nach dez, Ausgabe in
                                                 ASCII
24946  a838
24947  a838  68                 pla              yr
24948  a839  a8                 tay
24949  a83a
24950  a83a  60                 rts
24951  a83b
24952  a83b
```

```
24953  a83b  ===> High-Byte Zeilennr./Direktmod. eintr. <===
24954  a83b
24955  a83b  85 3c      retpat sta curlin+1     High-Byte akt. Zeilennummer
24956  a83d  88                dey
24957  a83e
24958  a83e  ===> Test auf Direktmodus, eintragen <===
24959  a83e
24960  a83e  aa                tax             High-Byte akt. Zeilennummer
24961  a83f  e8                inx             +1
24962  a840  d0 02             bne retpa1      nicht Direktmodus
24963  a842
24964  a842  86 7f             stx runmod      Direktmodus (=0) in Modusflag
24965  a844
24966  a844  60         retpa1 rts
24967  a845
24968  a845
24969  a845
24969  a845  ===> I/O-Chips einblenden <===
24970  a845
24971  a845  48         mapio  pha             ac
24972  a846
24973  a846  a9 00             lda #$00
24974  a848
24975  a848   00xxxxxx  RAM Bank 0
24976  a848   xx00xxxx  KERNAL ROM       $c000-$cfff
24977  a848   xxxx00xx  BASIC ROM high   $8000-$bfff
24978  a848   xxxxxx0x  BASIC ROM low    $4000-$7fff
24979  a848   xxxxxxx0  I/O-Chips        $d000-$dfff
24980  a848
24981  a848  8d 00 ff          sta mmucr       MMU 8722 Config. Reg. (Kopie)
24982  a84b
24983  a84b  68                pla             ac
24984  a84c  60                rts
24985  a84d
24986  a84d
24987  a84d                    .end
24988  a84d                    .lib basic.irq
24989  a84d  put"@basic.irq"
```

```
24991  a84d
24992  a84d
24993  a84d  ===> BASIC-IRQ Sprites, Light-Pen, Sound <===
24994  a84d
24995  a84d  ad fd 12   basirq lda birqfg       BASIC-IRQ Flag (<>0 = in Arbeit)
24996  a850  f0 01             beq bsir10       BASIC-IRQ bearbeiten
24997  a852
24998  a852  60               rts
24999  a853
25000  a853
25001  a853  ===> BASIC-IRQ bearbeiten <===
25002  a853
25003  a853  ee fd 12   bsir10 inc birqfg       erneuten Aufruf BASIC-IRQ verhindern
25004  a856
25005  a856  ===> Aktualisieren Sprite-Koordinaten <===
25006  a856
25007  a856  a2 10             ldx #16          Zähler für Koordinaten 8 Sprites
25008  a858
25009  a858
25010  a858  ===> Bewegung/Kollision IRQ Sprites bearbeiten <===
25011  a858
25012  a858  bd d6 11   bsir15 lda postab,x     Abszisse Sprite
25013  a85b  9d 00 d0           sta vic,x       VIC 8564 Basisadresse
25014  a85e
25015  a85e  ca               dex
25016  a85f  10 f7             bpl bsir15      Bewegung/Kollision IRQ Sprites
                                               bearbeiten
25017  a861
25018  a861  ===> Test / neue Sprite Koordinaten berechnen <===
25019  a861
25020  a861  a0 07             ldy #7           Zähler für 8 Sprites
25021  a863
25022  a863
25023  a863  ===> Test auf Sprite y eingeschaltet <===
25024  a863
25025  a863  ad 15 d0   bsir20 lda vicmoe       VIC 8564 Sprite Enable Register
25026  a866
25027  a866   76543210  Sprite7-0 (1=eingeschaltet)
25028  a866
25029  a866  39 b3 6c           and sbits,y     Tabelle Maskenbits Bit0-7
25030  a869  f0 38             beq bsir35       Sprite y ausgeschaltet
25031  a86b
25032  a86b  ===> Test Bewegung Sprite y <===
25033  a86b
25034  a86b  be d9 6d           ldx sproff,y    Tabelle Offsets für Parameter
                                               Sprite0-7
25035  a86e  bd 7e 11           lda sspeed,x    Konstante Schritte / BASIC-IRQ
25036  a871  f0 30             beq bsir35       Sprite y steht still
25037  a873
25038  a873  9d 7f 11           sta scnst,x     IRQ-Zähler für Geschw. Sprite laden
25039  a876
25040  a876
25041  a876  ===> nächste Sprite-Position berechnen <===
25042  a876
25043  a876  98         bsir25 tya             Zähler für Sprite7-0
25044  a877  0a                asl a           mal zwei ergibt
25045  a878  a8                tay             Zeiger in Tab. Sprite-Pos.
25046  a879
```

```
25047  a879  bd 80 11          lda squad,x      Quadrant Sprite
25048  a87c  38                sec
25049  a87d  e9 01             sbc #1
25050  a87f
25051  a87f  e8                inx              Zeiger in Sprite-Daten
25052  a880  e8                inx              für Berechnung y-Wert
25053  a881
25054  a881  c8                iny              Zeiger auf y-Wert Sprite-Pos.
25055  a882
25056  a882  20 f4 a9          jsr nxscor       nächste Sprite-Koordinate berechnen
25057  a885
25058  a885  ca                dex              Zeiger in Sprite-Daten
25059  a886  ca                dex              für Berechnung x-Wert
25060  a887
25061  a887  88                dey              Zeiger auf x-Wert Sprite-Pos.
25062  a888
25063  a888  bd 80 11          lda squad,x      Quadrant Sprite
25064  a88b  20 f4 a9          jsr nxscor       nächste Sprite-Koordinate berechnen
25065  a88e
25066  a88e  08                php
25067  a88f  98                tya              Zeiger
25068  a890  4a                lsr a            dividiert durch 2
25069  a891  a8                tay              ergibt Zeiger in Sprite-Daten
25070  a892  28                plp
25071  a893
25072  a893  90 09             bcc bsir30       Sprite-Schritte zählen / Pos.
                                                 berechnen
25073  a895
25074  a895  ===> Übertrag x-Wert <===
25075  a895
25076  a895  ad e6 11          lda spmsb        MSB-Position Sprite7-0
25077  a898  59 b3 6c          eor sbits,y      Tabelle Maskenbits Bit0-7
25078  a89b  8d e6 11          sta spmsb        MSB-Position Sprite7-0
25079  a89e
25080  a89e
25081  a89e  ===> Sprite-Schritte zählen / Pos. berechnen <===
25082  a89e
25083  a89e  de 7f 11    bsir30 dec scnst,x     Zähler für Schritte /BASIC-IRQ
25084  a8a1  d0 d3             bne bsir25       nächste Sprite-Position berechnen
25085  a8a3
25086  a8a3
25087  a8a3  ===> Test / Bewegung nächstes Sprite <===
25088  a8a3
25089  a8a3  88          bsir35 dey             Zähler für Sprites
25090  a8a4  10 bd             bpl bsir20       Test auf Sprite y eingeschaltet
25091  a8a6
25092  a8a6  ad 19 d0          lda vicifr       VIC 8564 Interrupt Flag-Register
25093  a8a9  8d 19 d0          sta vicifr       alle IRQ-Flags rücksetzen
25094  a8ac
25095  a8ac   7.......  Interrupt-Flag           (1=VIC IRQ)
25096  a8ac   .654....  unbenutzt
25097  a8ac   ....3...  Light-Pen IRQ               (1=IRQ)
25098  a8ac   .....2..  Sprite/Sprite Kollision     (1=IRQ)
25099  a8ac   ......1.  Sprite/Hintergrund Kollision (1=IRQ)
25100  a8ac   .......0  Raster-Vergleich Interrupt  (1=IRQ)
25101  a8ac
25102  a8ac  29 0e             and #%00001110   Light-Pen / Kollisions-IRQ maskieren
25103  a8ae  f0 44             beq musirq       kein maskiertes Flag gesetzt
```

```
25104  a8b0
25105  a8b0  4a              lsr a           Raster-IRQ Flag verwerfen
25106  a8b1
25107  a8b1  a0 01           ldy #1          Zähler für Kollisions-IRQ
25108  a8b3
25109  a8b3
25110  a8b3  ===> Test nächstes Flag Koll.-IRQ <===
25111  a8b3
25112  a8b3  4a        bsir40 lsr a          Flag Kollisions-IRQ ins Carry
25113  a8b4  90 20            bcc bsir55      nicht gesetzt
25114  a8b6
25115  a8b6  48              pha             VIC 8564 IRQ-Flags
25116  a8b7
25117  a8b7  b9 1e d0         lda vicssc,y    Kollisionsflags nach ac
25118  a8ba
25119  a8ba   vicsbc    für yr=1:
25120  a8ba   76543210  Hintergrund-Kollision mit Sprite7-0
25121  a8ba
25122  a8ba   vicssc    für yr=0:
25123  a8ba   76543210  Sprite-Kollision mit Sprite7-0
25124  a8ba
25125  a8ba  19 e7 11         ora sscoll,y
25126  a8bd  99 e7 11         sta sscoll,y    Kopie Kollisionsflags
25127  a8c0
25128  a8c0   sbcoll    für yr=1:
25129  a8c0   76543210  Hintergrund-Kollision mit Sprite7-0
25130  a8c0
25131  a8c0   sscoll    für yr=0:
25132  a8c0   76543210  Sprite-Kollision mit Sprite7-0
25133  a8c0
25134  a8c0  a9 00            lda #%00000000  Kollisionsflags Sprite7-0
25135  a8c2  99 1e d0         sta vicssc,y    in VIC 8564 rücksetzen
25136  a8c5
25137  a8c5  ad 7f 12         lda intval      Flag für Bearbeitung COLLISION
25138  a8c8
25139  a8c8   7.......  1=COLLISION wird bearbeitet
25140  a8c8   .6543...  unbenutzt
25141  a8c8   .....2..  1=Zeilennr. für Light-Pen definiert
25142  a8c8   ......1.  1=Zeilennr. für Sprite/Hintergr. Koll. def.
25143  a8c8   .......0  1=Zeilennr. für Sprite/Sprite Koll. def.
25144  a8c8
25145  a8c8  c0 00            cpy #0          Zählerstand Sprite/Sprite
25146  a8ca  f0 01            beq bsir45
25147  a8cc
25148  a8cc  4a              lsr a           Bit0 intval verwerfen
25149  a8cd
25150  a8cd  4a        bsir45 lsr a          Kollisionsflag ins Carry
25151  a8ce  90 05            bcc bsir50      keine Zeilennummer definiert
25152  a8d0
25153  a8d0  a9 ff            lda #%11111111  Flag setzen
25154  a8d2  99 76 12         sta tripfg,y    für Bearbeitung COLLISION
25155  a8d5
25156  a8d5  68        bsir50 pla             VIC 8564 IRQ-Flags
25157  a8d6  88        bsir55 dey             Zähler für Koll. IRQ
25158  a8d7  10 da            bpl bsir40      Test nächstes Flag Koll.-IRQ
25159  a8d9
25160  a8d9  ===> Light-Pen IRQ bearbeiten <===
25161  a8d9
```

```
25162  a8d9  4a              lsr a           Flag für Light-Pen ins Carry
25163  a8da  90 18           bcc musirq      keine Zeilennr. Light-Pen definiert
25164  a8dc
25165  a8dc  ad 13 d0        lda viclpx      VIC 8564 Light-Pen x-Wert
25166  a8df  8d e9 11        sta lpabsz      Kopie Light-Pen, x-Wert (Abszisse)
25167  a8e2
25168  a8e2  ad 14 d0        lda viclpy      VIC 8564 Light-Pen y-Wert
25169  a8e5  8d ea 11        sta lpord       Kopie Light-Pen, y-Wert (Ordinate)
25170  a8e8
25171  a8e8  ad 7f 12        lda intval      Flag für Bearbeitung COLLISION
25172  a8eb
25173  a8eb   7.......  1=COLLISION wird bearbeitet
25174  a8eb   .6543...  unbenutzt
25175  a8eb   .....2..  1=Zeilennr. für Light-Pen definiert.
25176  a8eb   ......1.  1=Zeilennr. für Sprite/Hintergr. Koll. def.
25177  a8eb   .......0  1=Zeilennr. für Sprite/Sprite Koll. def.
25178  a8eb
25179  a8eb  29 04           and #%00000100  Light-Pen Bit2 maskieren
25180  a8ed  f0 05           beq musirq      keine Zeilennr. definiert
25181  a8ef
25182  a8ef  a9 ff           lda #%11111111  Flag setzen für
25183  a8f1  8d 78 12        sta lpnflg      COLLISION Light-Pen
25184  a8f4
25185  a8f4
25186  a8f4  ===> PLAY Tondauer bearbeiten <===
25187  a8f4
25188  a8f4  a2 00     musirq ldx #0         Zähler für (3) Stimmen
25189  a8f6  bd 24 12  msir10 lda voichi,x   Tab. High-Bytes Tondauer (1-3)
25190  a8f9  30 27            bmi msir15
25191  a8fb
25192  a8fb  bd 23 12         lda voiclo,x   Tab. Low-Bytes Tondauer (1-3)
25193  a8fe  38               sec
25194  a8ff  ed 22 12         sbc tmprat     Spiel Tempo
25195  a902  9d 23 12         sta voiclo,x   Tab. Low-Bytes Tondauer (1-3)
25196  a905
25197  a905  b0 1b            bcs msir15
25198  a907
25199  a907  bd 24 12         lda voichi,x   Tab. High-Bytes Tondauer (1-3)
25200  a90a  e9 00            sbc #$00
25201  a90c  9d 24 12         sta voichi,x   Tab. High-Bytes Tondauer (1-3)
25202  a90f
25203  a90f  b0 11            bcs msir15
25204  a911
25205  a911  8a               txa
25206  a912  4a               lsr a
25207  a913  a8               tay
25208  a914  b9 30 12         lda wave,y     Tab. HÜllk.werte VOICE 1-3
25209  a917  29 fe            and #$fe
25210  a919  48               pha
25211  a91a
25212  a91a  b9 39 70         lda offtab,y   Tab. SID Offsets VOICE 1-3
25213  a91d  a8               tay
25214  a91e  68               pla
25215  a91f  99 04 d4         sta sid+osc1+oscctl,y  SID Stimme 1 WELLENFORM
25216  a922
25217  a922  e8        msir15 inx
25218  a923  e8               inx
25219  a924  e0 06            cpx #$06
```

```
25220  a926  d0 ce           bne msir10
25221  a928
25222  a928  a0 02           ldy #$02
25223  a92a  b9 85 12  msir20 lda sndth1,y     High-Byte Tondauer VOICE 1
25224  a92d  10 06           bpl msir30
25225  a92f
25226  a92f  88        msir25 dey
25227  a930  10 f8           bpl msir20
25228  a932
25229  a932  4c f0 a9        jmp irqend
25230  a935
25231  a935
25232  a935  18        msir30 clc
25233  a936  b9 9d 12         lda sdfql1,y     Low-Byte Frequenz VOICE 1
25234  a939  79 97 12         adc sdspl1,y     Low-Byte Schrittweite VOICE 1
25235  a93c  99 9d 12         sta sdfql1,y     Low-Byte Frequenz VOICE 1
25236  a93f
25237  a93f  b9 a0 12         lda sdfqh1,y     High-Byte Frequenz VOICE 1
25238  a942  79 9a 12         adc sdsph1,y     High-Byte Schrittweite VOICE 1
25239  a945  99 a0 12         sta sdfqh1,y     High-Byte Frequenz VOICE 1
25240  a948
25241  a948  b9 94 12         lda sddir1,y     Richtung VOICE 1
25242  a94b  aa               tax
25243  a94c  29 01            and #$01
25244  a94e  f0 2e            beq msir45
25245  a950
25246  a950  90 0f            bcc msir35
25247  a952
25248  a952  38               sec
25249  a953  b9 9d 12         lda sdfql1,y     Low-Byte Frequenz VOICE 1
25250  a956  f9 8e 12         sbc sdmnl1,y     Low-Byte Grenzfreq.2 VOICE 1
25251  a959  b9 a0 12         lda sdfqh1,y     High-Byte Frequenz VOICE 1
25252  a95c  f9 91 12         sbc sdmnh1,y     High-Byte Grenzfreq.2 VOICE 1
25253  a95f  b0 4d            bcs msir60
25254  a961
25255  a961  e0 02     msir35 cpx #$02
25256  a963  90 0a            bcc msir40
25257  a965
25258  a965  20 da a9         jsr negstp
25259  a968  a9 02            lda #$02
25260  a96a  99 94 12         sta sddir1,y     Richtung VOICE 1
25261  a96d
25262  a96d  d0 33            bne msir55
25263  a96f
25264  a96f  b9 88 12  msir40 lda sdmxl1,y     Low-Byte Grenzfreq.1 VOICE 1
25265  a972  99 9d 12         sta sdfql1,y     Low-Byte Frequenz VOICE 1
25266  a975
25267  a975  b9 8b 12         lda sdmxh1,y     High-Byte Grenzfreq.1 VOICE 1
25268  a978  99 a0 12         sta sdfqh1,y     High-Byte Frequenz VOICE 1
25269  a97b
25270  a97b  4c ae a9         jmp msir60
25271  a97e
25272  a97e
25273  a97e  b0 14     msir45 bcs msir50
25274  a980
25275  a980  b9 a0 12         lda sdfqh1,y     High-Byte Frequenz VOICE 1
25276  a983  d9 8b 12         cmp sdmxh1,y     High-Byte Grenzfreq.1 VOICE 1
25277  a986  90 26            bcc msir60
```

```
25278  a988
25279  a988  d0 0a           bne msir50
25280  a98a
25281  a98a  b9 9d 12        lda sdfql1,y     Low-Byte Frequenz VOICE 1
25282  a98d  d9 88 12        cmp sdmxl1,y     Low-Byte Grenzfreq.1 VOICE 1
25283  a990  90 1c           bcc msir60
25284  a992
25285  a992  f0 1a           beq msir60
25286  a994
25287  a994  e0 02     msir50 cpx #$02
25288  a996  90 0a           bcc msir55
25289  a998
25290  a998  20 da a9        jsr negstp
25291  a99b  a9 03           lda #$03
25292  a99d  99 94 12        sta sddir1,y     Richtung VOICE 1
25293  a9a0
25294  a9a0  d0 cd           bne msir40
25295  a9a2
25296  a9a2  b9 8e 12  msir55 lda sdmnl1,y     Low-Byte Grenzfreq.2 VOICE 1
25297  a9a5  99 9d 12        sta sdfql1,y     Low-Byte Frequenz VOICE 1
25298  a9a8
25299  a9a8  b9 91 12        lda sdmnh1,y     High-Byte Grenzfreq.2 VOICE 1
25300  a9ab  99 a0 12        sta sdfqh1,y     High-Byte Frequenz VOICE 1
25301  a9ae
25302  a9ae  be 39 70  msir60 ldx offtab,y     Tab. SID Offsets VOICE 1-3
25303  a9b1  b9 9d 12        lda sdfql1,y     Low-Byte Frequenz VOICE 1
25304  a9b4  9d 00 d4        sta sid+freqlo,x SID Low-Byte Osz.Frequ.
25305  a9b7
25306  a9b7  b9 a0 12        lda sdfqh1,y     High-Byte Frequenz VOICE 1
25307  a9ba  9d 01 d4        sta sid+freqhi,x SID High-Byte Osz.Frequ.
25308  a9bd
25309  a9bd  98              tya
25310  a9be  aa              tax
25311  a9bf  bd 82 12        lda sndtl1,x     Low-Byte Tondauer VOICE 1
25312  a9c2  d0 03           bne msir65
25313  a9c4
25314  a9c4  de 85 12        dec sndth1,x     High-Byte Tondauer VOICE 1
25315  a9c7  de 82 12  msir65 dec sndtl1,x     Low-Byte Tondauer VOICE 1
25316  a9ca  bd 85 12        lda sndth1,x     High-Byte Tondauer VOICE 1
25317  a9cd  10 08           bpl msir70
25318  a9cf
25319  a9cf  a9 08           lda #$08
25320  a9d1  be 39 70        ldx offtab,y     Tab. SID Offsets VOICE 1-3
25321  a9d4  9d 04 d4        sta sid+osc1+osccctl,x  SID Stimme 1 WELLENFORM
25322  a9d7
25323  a9d7  4c 2f a9  msir70 jmp msir25
25324  a9da
25325  a9da
25326  a9da  b9 97 12  negstp lda sdspl1,y     Low-Byte Schrittweite VOICE 1
25327  a9dd  49 ff           eor #$ff
25328  a9df  18              clc
25329  a9e0  69 01           adc #$01
25330  a9e2  99 97 12        sta sdspl1,y     Low-Byte Schrittweite VOICE 1
25331  a9e5
25332  a9e5  b9 9a 12        lda sdsph1,y     High-Byte Schrittweite VOICE 1
25333  a9e8  49 ff           eor #$ff
25334  a9ea  69 00           adc #$00
25335  a9ec  99 9a 12        sta sdsph1,y     High-Byte Schrittweite VOICE 1
```

```
25336  a9ef
25337  a9ef  60              rts
25338  a9f0
25339  a9f0
25340  a9f0  ce fd 12  irqend dec birqfg      BASIC-IRQ Flag (<>0 = in Arbeit)
25341  a9f3  60              rts
25342  a9f4
25343  a9f4
25344  a9f4  ===> nächste Sprite-Koordinate berechnen <===
25345  a9f4
25346  a9f4  48        nxscor pha             Quadrant-1
25347  a9f5
25348  a9f5  In Abhängigkeit vom xr wird die Abszisse (x-Wert)
25349  a9f5  oder die Ordinate (y-Wert) der nächsten Position
25350  a9f5  eines Sprites berechnet.
25351  a9f5
25352  a9f5  18              clc
25353  a9f6  bd 81 11        lda sxswt,x      Low-Byte letzte Koordinate
25354  a9f9  7d 85 11        adc sabsz,x      plus Low-Byte Schrittweite
25355  a9fc  9d 85 11        sta sabsz,x      gleich Low-Byte neue Koordinate
25356  a9ff
25357  a9ff  bd 82 11        lda sxswt+1,x    High-Byte letzte Koordinate
25358  aa02  7d 86 11        adc sabsz+1,x    plus High-Byte Schrittweite
25359  aa05  9d 86 11        sta sabsz+1,x    gleich High-Byte neue Koordinate
25360  aa08
25361  aa08  68              pla              Quadrant-1
25362  aa09  90 13           bcc nxsc30
25363  aa0b
25364  aa0b  ===> Wechsel Quadrant <===
25365  aa0b
25366  aa0b  4a              lsr a
25367  aa0c  4a              lsr a            Bit2 ins Carry
25368  aa0d
25369  aa0d  b9 d6 11        lda postab,y     Abszisse Sprite
25370  aa10  b0 05           bcs nxsc10
25371  aa12
25372  aa12  69 01           adc #$01
25373  aa14  4c 1b aa        jmp nxsc20
25374  aa17
25375  aa17
25376  aa17  e9 01     nxsc10 sbc #1
25377  aa19  c9 ff            cmp #255
25378  aa1b  99 d6 11  nxsc20 sta postab,y     Abszisse Sprite
25379  aa1e
25380  aa1e  60        nxsc30 rts
25381  aa1f
25382  aa1f
25383  aa1f  ===> BASIC-Statement STASH ($fe $1f) <===
25384  aa1f
25385  aa1f  a9 84     stash  lda #$84
25386  aa21  4c 2b aa         jmp swap01
25387  aa24
25388  aa24
25389  aa24  ===> BASIC-Statement FETCH ($fe $21) <===
25390  aa24
25391  aa24  a9 85     fetch  lda #$85
25392  aa26  4c 2b aa         jmp swap01
25393  aa29
```

```
25394  aa29
25395  aa29  ===> BASIC-Statement SWAP ($fe $23) <===
25396  aa29
25397  aa29  a9 86      swap   lda #$86
25398  aa2b  48         swap01 pha
25399  aa2c
25400  aa2c  20 12 88          jsr getwrd       16 Bit INT nach linnum
25401  aa2f  20 45 a8          jsr mapio        I/O-Chips einblenden
25402  aa32  8c 07 df          sty dma+7
25403  aa35
25404  aa35  8d 08 df          sta dma+8
25405  aa38
25406  aa38  20 0f 88          jsr comwrd       Kommatest, 16 Bit INT nach linnum
25407  aa3b  20 45 a8          jsr mapio        I/O-Chips einblenden
25408  aa3e  8c 02 df          sty dma+2
25409  aa41
25410  aa41  8d 03 df          sta dma+3
25411  aa44
25412  aa44  20 0f 88          jsr comwrd       Kommatest, 16 Bit INT nach linnum
25413  aa47  20 45 a8          jsr mapio        I/O-Chips einblenden
25414  aa4a  8c 04 df          sty dma+4
25415  aa4d
25416  aa4d  8d 05 df          sta dma+5
25417  aa50
25418  aa50  20 09 88          jsr combyt       Kommatest, 8 Bit INT nach xr
25419  aa53  e0 10             cpx #$10
25420  aa55  b0 0e             bcs swaper       Behandlung »?illegal quantity error«
25421  aa57
25422  aa57  20 45 a8          jsr mapio        I/O-Chips einblenden
25423  aa5a  8e 06 df          stx dma+6
25424  aa5d
25425  aa5d  68                pla
25426  aa5e  a8                tay
25427  aa5f  ae d5 03          ldx curbnk       akt. Bank
25428  aa62  4c 50 ff          jmp kdmacl       DMA-Anforderung ext. RAM zulassen
                                                (Sprungv.)
25429  aa65
25430  aa65
25431  aa65  4c 28 7d   swaper jmp fcerr        Behandlung »?illegal quantity error«
25432  aa68
25433  aa68
25434  aa68  ===> GK in FAC runden, nach INT <===
25435  aa68
25436  aa68  20 47 8c   rndqnt jsr round       FAC runden
25437  aa6b  4c c7 8c          jmp qint         FAC GK nach INT
25438  aa6e
25439  aa6e
25440  aa6e            *       = $ae63
25441  ae63
25442  ae63
25443  ae63  Die folgenden Bytes enthalten verschlüsselt den Text:
25444  ae63
25445  ae63       Brought to you by...
25446  ae63
25447  ae63  Software:
25448  ae63   Fred Bowen
25449  ae63   Terry Ryan
25450  ae63   von Ertwine
```

```
25451  ae63
25452  ae63  Herdware:
25453  ae63   Bil Herd
25454  ae63   Dave Haynie
25455  ae63   Frank Palaia
25456  ae63
25457  ae63  Link arms, don't make them.
25458  ae63
25459  ae63
25460  ae63  Der Text kann durch (z.B.) folgende Eingaben
25461  ae63  im Direktmodus auf den Bildschirm gebracht
25462  ae63  werden:
25463  ae63
25464  ae63  poke 4624,255: poke 4625,255: poke 4626,68
25465  ae63  poke 4627,59 : poke 53,44: print fre(0) <ret>
25466  ae63
25467  ae63
25468  ae63  ===> verschlüsselte Mitteilung PGM-Autoren <===
25469  ae63
25470  ae63
25471  ae63  c7 45 57   kydmsg .byte $c7, $45, $57, $49, $44, $4d
25471  ae66  49 44 4d
25472  ae69  45 54 20          .byte $45, $54, $20, $4d, $45, $49
25472  ae6c  4d 45 49
25473  ae6f  4e 45 52          .byte $4e, $45, $52, $20, $c6, $52
25473  ae72  20 c6 52
25474  ae75  41 55 20          .byte $41, $55, $20, $c7, $41, $42
25474  ae78  c7 41 42
25475  ae7b  52 49 45          .byte $52, $49, $45, $4c, $45, $2c
25475  ae7e  4c 45 2c
25476  ae81  20 55 4e          .byte $20, $55, $4e, $53, $45, $52
25476  ae84  53 45 52
25477  ae87  45 52 20          .byte $45, $52, $20, $d4, $4f, $43
25477  ae8a  d4 4f 43
25478  ae8d  48 54 45          .byte $48, $54, $45, $52, $20, $cd
25478  ae90  52 20 cd
25479  ae93  41 47 44          .byte $41, $47, $44, $41, $4c, $45
25479  ae96  41 4c 45
25480  ae99  4e 41 2c          .byte $4e, $41, $2c, $20, $55, $4e
25480  ae9c  20 55 4e
25481  ae9f  53 45 52          .byte $53, $45, $52, $45, $4d, $20
25481  aea2  45 4d 20
25482  aea5  c8 55 4e          .byte $c8, $55, $4e, $44, $20, $c1
25482  aea8  44 20 c1
25483  aeab  4e 59 41          .byte $4e, $59, $41, $2c, $20, $44
25483  aeae  2c 20 44
25484  aeb1  49 45 20          .byte $49, $45, $20, $57, $5b, $48
25484  aeb4  57 5b 48
25485  aeb7  52 45 4e          .byte $52, $45, $4e, $44, $20, $44
25485  aeba  44 20 44
25486  aebd  45 52 20          .byte $45, $52, $20, $38, $30, $30
25486  aec0  38 30 30
25487  aec3  20 d3 54          .byte $20, $d3, $54, $55, $4e, $44
25487  aec6  55 4e 44
25488  aec9  45 4e 20          .byte $45, $4e, $20, $c1, $52, $42
25488  aecc  c1 52 42
25489  aecf  45 49 54          .byte $45, $49, $54, $20, $41, $4c
25489  aed2  20 41 4c
```

```
25490  aed5  4c 45 20          .byte $4c, $45, $20, $5a, $55, $20
25490  aed8  5a 55 20
25491  aedb  4b 55 52          .byte $4b, $55, $52, $5a, $20, $47
25491  aede  5a 20 47
25492  aee1  45 4b 4f          .byte $45, $4b, $4f, $4d, $4d, $45
25492  aee4  4d 4d 45
25493  aee7  4e 20 53          .byte $4e, $20, $53, $49, $4e, $44
25493  aeea  49 4e 44
25494  aeed  2e 20 32          .byte $2e, $20, $32, $35, $2e, $30
25494  aef0  35 2e 30
25495  aef3  35 2e 38          .byte $35, $2e, $38, $36, $20, $2d
25495  aef6  36 20 2d
25496  aef9  4f 4d 42          .byte $4f, $4d, $42, $2d, $2a, $2a
25496  aefc  2d 2a 2a
25497  aeff  2a                .byte $2a
25498  af00
25499  af00
25500  af00                    .end
25501  af00                    .lib basic.jmptbl
25502  af00  put"@basic.jmptbl"
```

```
25504  af00
25505  af00
25506  af00  ===> Sprungverteiler BASIC-Routinen <===
25507  af00
25508  af00  4c b4 84          jmp ayint       Wandlung FAC nach sgnd. INT
25509  af03  4c 3c 79          jmp givayv      16 Bit INT yr/ac nach GK in FAC
25510  af06  4c 42 8e          jmp fout        FAC nach String, Adr. nach ac/yr
25511  af09  4c 52 80          jmp val1        String auswerten
25512  af0c  4c 15 88          jmp getadr      pos. FAC als 16 Bit INT linnum,
                                                yr/ac
25513  af0f  4c 75 8c          jmp floatc      EX nach FAC, Nullen auffüllen,
                                                normal.
25514  af12  4c 2e 88          jmp fsub        RAM Var. ac/yr minus FAC
25515  af15  4c 31 88          jmp fsubt       BASIC-Funktion MINUS ($ab)
25516  af18  4c 45 88          jmp fadd        FAC plus Var. ac/yr
25517  af1b  4c 48 88          jmp faddt       BASIC-Funktion PLUS ($aa)
25518  af1e  4c 24 8a          jmp fmult       FAC mal RAM VAR ac/yr
25519  af21  4c 27 8a          jmp fmultt      BASIC-Funktion MULTIPLIKATION ($ac)
25520  af24  4c 49 8b          jmp fdiv        RAM Var. ac/yr durch FAC dividieren
25521  af27  4c 4c 8b          jmp fdivt       BASIC-Funktion DIVISION / ($ad)
25522  af2a  4c ca 89          jmp log         BASIC-Funktion LOG ($bc)
25523  af2d  4c fb 8c          jmp int         BASIC-Funktion INT ($b5)
25524  af30  4c b7 8f          jmp sqr         BASIC-Funktion SQR ($ba)
25525  af33  4c fa 8f          jmp negop       BASIC-Funktion NEG. VORZEICHEN
25526  af36  4c be 8f          jmp fpwr
25527  af39  4c c1 8f          jmp fpwrt       BASIC-Funktion POTENZ  ($ae)
25528  af3c  4c 33 90          jmp exp         BASIC-Funktion EXP ($bd)
25529  af3f  4c 09 94          jmp cos         BASIC-Funktion COS ($be)
25530  af42  4c 10 94          jmp sin         BASIC-Funktion SIN ($bf)
25531  af45  4c 59 94          jmp tan         BASIC-Funktion TAN ($c0)
25532  af48  4c b3 94          jmp atn         BASIC-Funktion ATN ($c1)
25533  af4b  4c 47 8c          jmp round       FAC runden
25534  af4e  4c 84 8c          jmp abs         BASIC-Funktion ABS ($b6)
25535  af51  4c 57 8c          jmp sign        Flag für FAC NULL, PLUS, MINUS nach
                                                ac
25536  af54  4c 87 8c          jmp fcomp       FAC mit Var. ac/yr vergleichen
25537  af57  4c 37 84          jmp rnd0
25538  af5a  4c b4 8a          jmp conupk      RAM Var. ac/yr nach ARG
25539  af5d  4c 89 8a          jmp romupk      ROM Konst. ac/yr nach ARG
25540  af60  4c 85 7a          jmp movfrm
25541  af63  4c d4 8b          jmp movfm       RAM Var. ac/yr nach FAC
25542  af66  4c 00 8c          jmp movmf       FAC runden, nach RAM xr/yr
25543  af69  4c 28 8c          jmp movfa       ARG nach FAC
25544  af6c  4c 38 8c          jmp movaf       FAC runden, FAC nach ARG
25545  af6f  4c 28 48          jmp optab       Adresse / Prioritätsflags der math.
                                                Routinen
25546  af72  4c 30 9b          jmp drawln      Strecke zeichnen
25547  af75  4c fb 9b          jmp gplot       Test/einfachen/dopp. Punkt setzen
25548  af78  4c 50 67          jmp cirsub      Drehung ausführen
25549  af7b  4c 9b 5a          jmp run         BASIC-Statement RUN ($8a)
25550  af7e  4c f3 51          jmp runc        BASIC-Zeiger initialisieren, CLR
25551  af81  4c f8 51          jmp clear       BASIC-Statement CLR ($9c)
25552  af84  4c d6 51          jmp new         BASIC-Statement NEW ($a2)
25553  af87  4c 4f 4f          jmp lnkprg      alle Linkadressen neu berechnen
25554  af8a  4c 0a 43          jmp crunch      ind. Wandlung Text/Tokens
25555  af8d  4c 64 50          jmp fndlin      Adr. linnum/+1 nach lowtr/+1
25556  af90  4c f6 4a          jmp newstt      Stopptaste abfragen, nächst.
                                                BASIC-Befehl/Zeile
```

```
25557  af93  4c d7 78        jmp eval        ind. folg. Ausdruck auswerten
25558  af96  4c ef 77        jmp frmevl      folgenden Ausdruck auswerten
25559  af99  4c a6 5a        jmp runprg      RUN akt. PGM
25560  af9c  4c 81 5a        jmp setexc      PGM Modus setzen
25561  af9f  4c a0 50        jmp linget      Zeilennr. nach linnum
25562  afa2  4c ea 92        jmp garba2      Garbage Collection
25563  afa5  4c cd 4d        jmp execln
25564  afa8
25565  afa8
25566  afa8                  .end
25567  afa8
25568  afa8
25569  afa8                  .end

Fehler in Pass 1 = 00000
Fehler in Pass 2 = 00000

Assemblierung Ende
```

4 VERWEISTABELLE

 # = Definition, $ = Schreibzugriff, leer = Lesezugriff/Sprung

```
abs    = $8c84:  2842 , 18783#, 25534
absrts = $9dad: 22469 , 22490#
abstw2 = $9d9c: 22469#, 22502 , 22540
abstwo = $9d99:  8958 ,  9080 , 21920 , 22464#
add1   = $97e9: 21245 , 21256#
addind = $7e12: 15359 , 15369#
addoff = $a062: 23024 , 23031 , 23108#
addon  = $5292:  4992#,  5005 ,  5951 ,  6258
addtw2 = $9d70:  9554 , 22404 , 22420#
addtwo = $9d6d:  8905 ,  8970 , 22415#, 22773
adjust = $6a4c: 10323 , 10345 , 10359#
adof10 = $a06e: 23115 , 23127#
adray1 = $117a:   584#,  1706$,  1707$
adray2 = $117c:   585#,  1712$,  1713$
affrts = $5093:  4442 ,  4458#
afplus = $9853: 21318 , 21326#
afrm   = $982a: 21299#, 21382
ahp    = $96de: 21047 , 21050 , 21069#
alg    = $97fb: 20987 , 21270#
aline1 = $4f39:  4134#,  4139
aline2 = $4f44:  4135 ,  4141#
ana    = $95f2: 20921#, 20937
anaf   = $98f2: 20833 , 20899 , 20969 , 21427#
andop  = $4c89:  2902 ,  3581#
angcnt = $114e:   530#,  8836 ,  8996 ,  9003$,  9006$,  9641$,  9646$,  9647$,
                 9654$,  9655$, 21858$, 21861$
angd10 = $9af8: 21861#, 21871
angd20 = $9b0b: 21851 , 21873#
angdst = $9af3:  8985 ,  9568 ,  9634 , 21858#
angl10 = $676e:  9646#,  9668
angl20 = $678f:  9671#,  9674
anglpt = $6767:  9027 ,  9641#
angm10 = $9ad4: 21830 , 21834#
angm20 = $9aea: 21841 , 21849#
angm30 = $9aed: 21846 , 21850#
angmlt = $9ace: 11112 , 21829#, 21862 , 22726
angsgn = $1149:   527#,  8853 ,  8895 ,  8919$,  8924$,  8987 ,  8989$,  8999 , 11072
                21752$, 21763$, 21813 , 21834
angtab = $63f1:  9005 ,  9012#
```

```
angval = $9f29: 21787 , 21788 , 22824#
ans010 = $9794: 21190 , 21195#
ans020 = $9798: 21193 , 21198#
ans1   = $a7fd: 24884#, 24888
ans2   = $a806: 24885 , 24890#
ans3   = $a80a: 24874 , 24895#
ansub  = $978b: 21188#, 21428 , 21436 , 21458 , 21503 , 21513
append = $a134:  2811 , 23345#
areg   =   $06:    80#,  6430$,  6484
argexp =   $6a:   227#,  3620 ,  3621 , 14052$, 17757 , 17758 , 17807 , 17810 ,
                 18311$, 18350$, 18367 , 18674 , 18694$, 19533 , 19664 , 19668$
argho  =   $6b:   228#,  3617 ,  3618$, 14055$, 17903 , 18261 , 18307$, 18346$, 18491
                 18525$, 18546 , 18548$
arglo  =   $6e:   231#,  3640 ,  3643$, 14064$, 17891 , 18249 , 18288$, 18327$, 18503
                 18522$, 18534 , 18536$
argmo  =   $6d:   230#,  3639 ,  3642$,  3672 , 14061$, 17895 , 18253 , 18292$,
                 18331$, 18499 , 18523$, 18538 , 18540$
argmoh =   $6c:   229#, 14058$, 17899 , 18257 , 18296$, 18335$, 18495 , 18524$, 18542
                 18544$
argsgn =   $6f:   232#,  3615 , 14067$, 17727 , 17780 , 18300$, 18305 , 18339$, 18344
                 18666 , 19059 , 19543 , 20543
arisgn =   $70:   233#, 14070$, 15529$, 15538 , 17728$, 17803 , 18303$, 18342$, 18393
                 18427$, 18459$, 19061$, 19684$, 20548$
arydlp = $7c50: 14925#, 14929
arydon = $7c40: 14813 , 14912#
aryget = $7bfe: 14852 , 14858#, 14904 , 14907
arypnt =   $5a:   186#, 15247 , 15250$, 15254 , 15273$, 15276$, 15282$, 15398 , 15402
                 15478$, 15479$
arytab =   $31:   147#,  4881$,  4882$, 14650 , 14653 , 14767 , 14768 , 14793$,
                 14794$, 15099 , 15100
aryva2 = $7bba: 14802#, 14837 , 14840
aryva3 = $7bbe: 14808#, 14863
aryvgo = $7bc8: 14809 , 14812 , 14815#
asc    = $8677:  2858 , 17280#
aspac  = $55cf:  5731 ,  5755#
atkdcy =   $05:  1004#, 11507$, 12556$
atkmus = $7011:  1870 , 11860#
atktab = $123f:   639#,  1871$, 11506 , 12083 , 12219$
atn    = $94b3:  2853 , 20668#, 25532
atn1   = $94bb: 20670 , 20674#
atn2   = $94c9: 20678 , 20684#
atn3   = $94dc: 20690 , 20696#
atn4   = $94e2: 20697 , 20702#
atnc00 = $951b: 20767#
atnc01 = $9516: 20762#
atnc02 = $9511: 20757#
atnc03 = $950c: 20752#
atnc04 = $9507: 20747#
atnc05 = $9502: 20742#
atnc06 = $94fd: 20737#
atnc07 = $94f8: 20732#
atnc08 = $94f3: 20727#
atnc09 = $94ee: 20722#
atnc10 = $94e9: 20717#
atnc11 = $94e4: 20712#
atncon = $94e3: 20684 , 20685 , 20707#
attrib =   $f1:   349#, 10321 , 10328$, 12809 , 12813$, 12832$, 15999
```

```
autinc =   $74:   245#,  1731$,  1732$,  4106 ,  4107 ,  4114 ,  4118 ,  6639$,
                 6642$,  6891$,  6892$
auto   = $5975:  2764 ,  6634#
avf1   = $9941: 21468 , 21477#
ayint  = $84b4:  3584 ,  3595 ,  5324 , 14170 , 16833 , 16854#, 25508
ba     =   $c6:   323#
backup = $a37c:  2790 , 23804#
badco1 = $611e:  8439 ,  8446 ,  8455#
badcom = $4846:  2827 ,  2833 ,  2925#,  8455
badval = $69d5: 10158 , 10179 , 10202 , 10206 , 10213#
bank   = $6bc9:  2799 , 10760#
basirq = $a84d:  1647 , 24995#
baswrk = $1c01:   699#,  1785 ,  1786
begfd  = $0134:   442#, 21298 , 21432$, 21442 , 21450
bigges = $8e97: 19232 , 19255#
birqfg = $12fd:   696#,  1860$, 24995 , 25003$, 25340$
birqvc = $4006:  1647#
bitcnt = $1169:   569#,  9354$,  9370$,  9377$,  9470
bitidx = $1161:   567#,  8861 ,  9145$,  9174 ,  9197
bitmsk = $116d:   573#, 12885$, 12905 , 12911 , 22073$, 22076 , 22084 , 22086$, 22089
                22105$, 22108
bits   = $03df:   480#,  1725$,  4922$, 18023 , 18855$, 18866$, 18879$
blfd   = $0133:   441#, 20918$, 21313 , 21416
bload  = $a218:  2814 , 23521#
bload2 = $a234: 23535 , 23538 , 23541#
bload3 = $a25b: 23559 , 23566#
bload4 = $a265: 23571 , 23578#
blodnp = $a21f: 12715 , 23528#
bltlp  = $7c96: 15006#, 15010
bltu   = $7c66: 14787 , 14958#
bltu1  = $7c8a: 14989 , 14995#
bmperr = $8394: 16469 , 16485#
bnderr = $5377:  5127 ,  5133 ,  5199#
bnk    =   $02:    71#,  6424$, 12732$
bnkerr = $6bd4: 10762 , 10770#
bnr    = $0123:   425#, 20824$, 21107 , 21236 , 21244 , 21273 , 21281$, 21296
boot   = $7335:  2824 , 12702#
boot10 = $7351: 12725 , 12732#
boot20 = $735e: 12713 , 12745#
booter = $736f: 12716 , 12752 , 12759#
bout   = $9840: 21305 , 21313#, 21361 , 21406
box    = $62b7:  2769 ,  8792#
box05  = $62d7:  8810 ,  8817#
box10  = $62e4:  8829#,  8837
box15  = $62e7:  8826 ,  8834#
box20  = $62ef:  8839#,  8888
box30  = $62f1:  8840#,  8843
boxang = $1154:   540#,  8803$,  8804$,  8949
boxf05 = $6308:  8855 ,  8858#
boxf10 = $6318:  8866#,  8868
boxf20 = $631e:  8870#,  8944
boxf25 = $6320:  8871#,  8874
boxf28 = $632c:  8879#,  8883
boxf30 = $6335:  8885#,  8931
boxf40 = $633f:  8886 ,  8890#
boxf45 = $634e:  8897 ,  8901#
boxf50 = $6350:  8902#,  8927
boxf60 = $6361:  8911 ,  8915#
```

```
boxf65 = $636c:  8920 ,  8924#
boxf70 = $6378:  8933#,  8943
boxf80 = $6382:  8933 ,  8936 ,  8939#
boxfil = $62fe:  8825 ,  8851#
boxp10 = $6410:  9030#,  9045
boxp20 = $6424:  9035 ,  9038 ,  9041#
boxpnt = $640b:  8829 ,  8993 ,  8997 ,  9027#
boxs05 = $6453:  9080#,  9109
boxs10 = $646c:  9088 ,  9096#
boxs15 = $649c:  9125#,  9284
boxs20 = $64ac:  9132 ,  9137#
boxs25 = $64b1:  9127 ,  9141#
boxs30 = $64b6:  9142 ,  9145#
boxs35 = $64b9:  9147#,  9149 ,  9211 ,  9214
boxs40 = $64c7:  9157 ,  9160#
boxs45 = $64d3:  9164 ,  9167#
boxs50 = $64dc:  9169 ,  9172#
boxs55 = $64e1:  9175#,  9178
boxs60 = $64f6:  9188 ,  9193#
boxs65 = $650c:  9202 ,  9207#
boxs70 = $652e:  9223 ,  9228#
boxs75 = $6530:  9229#,  9237
boxs80 = $6567:  9217 ,  9220 ,  9266#
boxs85 = $6572:  9269 ,  9272#
boxsb1 = $6392:  8955#,  8982
boxsub = $6389:  8822 ,  8949#
boxtab = $63ed:  8992 ,  9001 ,  9010#
brkxit = $90d3: 19762#, 19822
bsav1  = $a1dd: 23475 , 23482#
bsav2  = $a1f1: 23485 , 23492#
bsave  = $a1c8:  2813 , 23464#
bserr  = $7d25: 15141#, 15173 , 15346
bserr7 = $7dfa: 15335 , 15346#
bsir10 = $a853: 24996 , 25003#
bsir15 = $a858: 25012#, 25016
bsir20 = $a863: 25025#, 25090
bsir25 = $a876: 25043#, 25084
bsir30 = $a89e: 25072 , 25083#
bsir35 = $a8a3: 25030 , 25036 , 25089#
bsir40 = $a8b3: 25112#, 25158
bsir45 = $a8cd: 25146 , 25150#
bsir50 = $a8d5: 25151 , 25156#
bsir55 = $a8d6: 25113 , 25157#
bsvrts = $a2a0: 23596 , 23609 , 23615 , 23623#
buf    = $0200:   450#,  4220$,  5696$,  5698 ,  5699 ,  5864 ,  5865 ,  5868$,
                  5924$,  5944 ,  6025$,  6027 ,  6028
bufful = $568d:  5929 ,  5936 ,  5944#
bump   = $837c:  2874 , 16465#
bup0   = $a38a: 23809 , 23816#
bup1   = $a390: 23817 , 23822#
bverck =   $0c:    94#, 19858$, 19867 , 19877 , 23441$
bvtrs  = $4267:  2004 ,  2005 ,  2022#
cat    = $870d: 13897 , 17422#
catlog = $a07e:  2782 ,  2809 , 23155#
cbasin = $9263: 19785 , 20108#, 23258 , 23264 , 23283 , 24836 , 24881 , 24887
cbn    = $97cd: 21232 , 21236#
```

```
cbsout = $9269:  1980 ,  1984 ,  8481 ,  8519 ,  8525 ,  8528 ,  8538 ,  8544 ,  8557
                 8567 , 10529 , 10548 , 12686 , 19778 , 20114#, 23278 , 23292 , 23299
                23644 , 23652
cckou1 = $90fc: 19800 , 19806#
cclall = $927b:  4855 , 20132#, 23594 , 23736
cclos  = $919a:  2753 , 19955#
cclose = $9275: 19960 , 20126#, 23319 , 23365 , 23404 , 23845 , 24856
cclrch = $926f:  3775 ,  5898 , 20120#, 23315 , 23356 , 23831 , 24853 , 24880
cdout  = $98eb: 20864 , 21063 , 21336 , 21350 , 21354 , 21379 , 21417 , 21422#
cenv3  =   $1c:  1088#
cff    = $9685: 20976 , 21012#
cform  = $0131:   439#, 20839 , 21199$, 21423$, 21454$, 21457$, 21531$
cfre1  = $938f: 20396 , 20402#
cfre2  = $9398: 20406 , 20411#
cfre3  = $93a7: 20418 , 20425#, 20428
cfre4  = $93b1: 20395 , 20399 , 20400 , 20435#
cfre4a = $93b3: 20437#, 20441 , 20450
cfre4b = $93c7: 20438 , 20455#
cgetl  = $9109:  6016 ,  6024 , 19820#
channl =   $15:   126#,  1659$,  1726$,  3778$,  5637$,  5700 ,  5709 ,  5807 ,
                 5858$,  5873 ,  5885$,  5893 ,  5901$,  5928 ,  5947 ,  5956 ,
                 6043 ,  6134 ,  6143 ,  6218
char   = $67d7:  2768 ,  9734#
char05 = $67e7:  9740 ,  9748#
char10 = $6803:  9765 ,  9772#
char15 = $680f:  9775 ,  9781#
char20 = $6815:  9778 ,  9784#
char25 = $6847:  9822 ,  9829#,  9837
char30 = $6858:  9830 ,  9842#
char35 = $6862:  9843 ,  9850#
char40 = $6863:  9810 ,  9855#
char45 = $6885:  9868 ,  9889#
char50 = $6890:  9873 ,  9876 ,  9881 ,  9884 ,  9891 ,  9898#
char55 = $689a:  9905#,  9940 ,  9947
char60 = $68b4:  9915 ,  9923#
char65 = $68bb:  9920 ,  9930#
char70 = $68c0:  9924 ,  9934#
char75 = $68c9:  9931 ,  9939#
char80 = $68d7:  9912 ,  9949#
charac =   $09:    84#,  3588$,  3605 ,  4489$,  4519 ,  5017$,  5023 ,  5025$,
                 6077$,  6083$,  6089$, 17317$, 17334
charer = $6800:  9756 ,  9769#
chargt = $5d9c:  7110 ,  7195 ,  7222 ,  7229 ,  7250 ,  7253 ,  7259 ,  7270 ,  7444
                 7529 ,  7535 ,  7572 ,  7608 ,  7630 ,  7635#
charom = $1168:   568#,  9858$,  9930$, 10005 , 12774$
chdgt1 = $427f:  2036 ,  2039#
chdgt2 = $4297:  2047 ,  2057#
chea3  = $4f65:  4163 ,  4169#
chead  = $4f58:  4161#,  4193
chgkey = $60e1:  8396#,  8442
chgky1 = $60ec:  8399 ,  8406#
chgtpt = $4279:  1717 ,  1718 ,  2004 ,  2035#,  2050
chgtx2 = $5d99:  7192 ,  7240 ,  7513 ,  7630#
chk1   = $a749: 23586 , 23629 , 24733#
chk2   = $a750: 23419 , 23446 , 23468 , 23528 , 24734 , 24743#
chk3   = $a75b: 23735 , 24754#
chk4   = $a760: 23770 , 23783 , 24762#
chk5   = $a766: 23795 , 24770#
```

```
chk6   = $a76f: 23327 , 23348 , 24780#
chkcls = $7956:  6567 , 14210#, 15086 , 16146 , 16256 , 16342 , 16381 , 16439 , 16465
                16502 , 16586 , 16607 , 16946 , 17229 , 21603 , 24417 , 24431
chkcom = $795c:  3466 ,  6191 ,  6397 ,  8408 ,  9781 , 10198 , 13589 , 14222#, 16379
                16500 , 17198 , 17675 , 17681 , 20013 , 22567 , 22595 , 22661 , 22682
chkds                chkds  = $79e3: 14315 , 14354#, 14442
chker1 = $a74d: 24738#, 24746 , 24755 , 24763 , 24772 , 24782
chkerr = $77e7:  6523 ,  9065 , 13773 , 13785#
chkesc = $5c40:  7294 ,  7309#
chkgrb = $9383: 20302 , 20370 , 20394#
chkk   = $996a: 21503#, 21510
chknum = $77da:  3517 ,  3555 ,  7774 , 13760#, 13928 , 14043 , 15691 , 15808 , 15865
                16576 , 16847 , 16945 , 16980 , 16994 , 17089
chkok  = $77e4: 13775#, 13780
chkopn = $7959:  3464 ,  3531 ,  6528 , 14204 , 14216#, 16331 , 16939 , 24415 , 24429
chkqut = $5c13:  7264#,  7339 ,  7377
chkstr = $77dd:  3467 ,  6546 ,  6577 , 13766#, 17429 , 17520 , 20783 , 21584
chkval = $77de:  3612 ,  5311 , 13761 , 13771#
cho    = $97ff: 21031 , 21273#
chom   = $999b: 21517 , 21535#
chou   = $9823: 21293 , 21296#
chout  = $981c: 21009 , 21292#
chrd   = $85bf:  2859 , 17102#
chrd1  = $85d1: 15781 , 15835 , 17118#
chrdsp = $68db:  9935 ,  9956#, 12792 , 12801 , 12821 , 12845 , 12932 , 13273
chrget = $0380:   466#,  1719$,  2219 ,  2264 ,  2325 ,  2332 ,  2381 ,  3144 ,  3168
                 3314 ,  3325 ,  3331 ,  3450 ,  3522 ,  3530 ,  3844 ,  4525 ,  5079
                 5085 ,  5108 ,  5109 ,  5117 ,  5181 ,  5219 ,  5275 ,  5767 ,  5846
                 5852 ,  6058 ,  6293 ,  6515 ,  6818 ,  6825 ,  7299 ,  7312 ,  7321
                 7338 ,  7352 ,  7387 ,  7417 ,  7487 ,  7813 ,  7977 ,  8105 ,  8249
                 8390 ,  8444 ,  9712 , 10641 , 11252 , 13874 , 14089 , 14118 , 14233
                14565 , 14576 , 14622 , 16330 , 16841 , 17648 , 19118 , 20781 , 20912
                22704 , 22753 , 24001 , 24055 , 24103 , 24106 , 24112 , 24202 , 24213
                24219 , 24304 , 24411 , 24425
chrglu = $d800:   715#,  1836
chrgot = $0386:   467#,  2213 ,  3851 ,  5044 ,  5059 ,  5090 ,  5111 ,  5651 ,  5996
                 6120 ,  6186 ,  6259 ,  6289 ,  6407 ,  6468 ,  6512 ,  6711 ,  6940
                 7024 ,  7504 ,  7807 ,  7858 ,  7969 ,  8028 ,  8055 ,  8172 ,  8191
                 8239 ,  9686 ,  9694 ,  9700 ,  9774 , 10118 , 10846 , 11219 , 11242
                13837 , 14536 , 14550 , 15080 , 17052 , 17194 , 17664 , 20003 , 20018
                20893 , 20905 , 21595 , 22564 , 22592 , 22614 , 22658 , 22690 , 22789
                23925 , 24177 , 24280
chrgrt = $5da4:  7637 ,  7639#
chrgug = $d000:   714#,  1833 , 12773
chrlop = $6185:  8556#,  8561
chsn   = $012b:   433#, 20834 , 20844 , 21481$
chx    = $9589: 20851#, 20865
cia1   = $dc00:  1422#
circ05 = $66c2:  9507 ,  9517 ,  9522#
circ10 = $66f9:  9548 ,  9560#
circ20 = $66fb:  9562#,  9565
circ30 = $670b:  9572#,  9575
circ40 = $6725:  9585 ,  9592#
circ45 = $6729:  9599#,  9613
circ50 = $6737:  9603 ,  9606#
circle = $668e:  2770 ,  9481#
cirend = $6748:  9609 ,  9616#
cirs10 = $6752:  9617 ,  9624#
```

```
cirs20 = $6757:  9628#,  9631
cirsub = $6750:  9577 ,  9611 ,  9623#, 25548
cl40_1 = $6afa: 10543 , 10547#
cl40_2 = $6b05: 10551 , 10554#
cl80_1 = $6ae6: 10524 , 10528#
cl80_2 = $6af1: 10532 , 10536#
clct2  = $a343: 23745 , 23749#
cld20  = $915f: 19887 , 19893#, 19927 , 19963
cld50  = $9160: 19878 , 19896#
cld60  = $916c: 19908 , 19922#
cld70  = $9184: 19926 , 19937#
clear  = $51f8:  2749 ,  4850#, 25551
clearc = $51fa:  4855#,  6939
clh010 = $a027: 23050 , 23056#
clh020 = $a037: 23073#, 23077 , 23086
clr40  = $6af2: 10457 , 10469 , 10541#
clr80  = $6ade: 10422 , 10427 , 10446 , 10522#
clrfac = $8d18: 18846 , 18909#
clrflg = $6ef7: 11464 , 11477 , 11488 , 11518 , 11546#, 11553
clrgf1 = $6b43: 10612 , 10616#
clrgf2 = $6b51: 10624#, 10628
clrgrf = $6b30: 10507 , 10604#, 12779 , 12863
clrhcl = $6b17: 10505 , 10577#
clrhir = $a022: 10640 , 23045#
clrplf = $6fce: 11275 , 11741 , 11766#
clrpu  = $522c:  4890#,  4893
clsn10 = $7189: 12260 , 12266#
clsner = $718d: 12241 , 12272#
cmd    = $5540:  2750 ,  5624 ,  5630#
cmdstr = $767f: 13006 , 13476#
cmo    = $9802: 21274#, 21286 , 21338 , 21342
cncj   = $9548: 20808#, 20814
coin   = $90fd:  5859 ,  5887 , 19811#
col00a = $931d: 20308#, 20372
col01  = $9329: 20303 , 20320#
col02  = $9333: 20331#, 20373
col02b = $935c: 20363#, 20366
col03  = $936f: 20321 , 20378#
colbt1 = $7625: 13383 , 13404#
colbt2 = $762a: 13389 , 13407#
colbt3 = $7634: 13394 , 13399 , 13405 , 13414#, 13527
colbyt = $7610: 13036 , 13357 , 13380#
colcnt = $115e:   561#,  9748$,  9755 ,  9758$,  9815 ,  9934 ,  9937$,  9943$,
                 12963$, 13039 , 13049 , 13138 , 13140$, 13205 , 13211$, 13227$,
                 13262$, 13446
coldvc = $4000:  1641#
colect = $a32f:  2787 , 23734#
colisn = $7164:  2820 , 12232#
colnum = $76b5: 12983 , 13505#
color  = $69e2:  2775 , 10234#
colr10 = $6a04: 10271 , 10282#
colr20 = $6a08: 10272 , 10286#
colr30 = $6a12: 10287 , 10298#
colr40 = $6a16: 10288 , 10302#
colr50 = $6a21: 10303 , 10314#
colr60 = $6a2d: 10315 , 10328#
colr70 = $6a32: 10304 , 10334#
colr80 = $6a43: 10277 , 10283 , 10293 , 10299 , 10309 , 10329 , 10348#
```

```
colrmh = $d800:   716#, 10591 , 22200
colrml = $1c00:   701#, 10618 , 22235 , 22236 , 22237 , 22238 , 22239 , 22240 , 22241
                22242 , 22243 , 22244 , 22245 , 22246 , 22247 , 22248 , 22249 , 22250
                22251 , 22252 , 22253 , 22254 , 22255 , 22256 , 22257 , 22258 , 22259
colsel =   $83:   289#,  8614 ,  9417$,  9684$,  9864 , 10033 , 22083 , 22095 , 22120
                22149 , 22636$
coltyp = $1280:   660#, 12243$, 12250
colval = $9f25: 13086 , 22085 , 22816#
com1   = $99af: 21439 , 21551#
com2   = $99b3: 21477 , 21554#
combyt = $8809:  6550 ,  6561 ,  9753 ,  9762 , 10251 , 10849 , 12615 , 12626 , 12645
                17675#, 21601 , 25418
comchk = $6dc6: 11201 , 11242#
comck1 = $6dd1: 11247 , 11250#
comck2 = $6dd8: 11243 , 11255#
comp   = $99a7: 21429 , 21545#
compar = $5dee:  7678 ,  7698 ,  7706#
comprt = $55a9:  5674 ,  5721#
comtop = $0400:   485#,  1791 ,  1792
comwrd = $880f: 12387 , 12397 , 16585 , 17681#, 23679 , 25406 , 25412
con1   = $97da: 21244#, 21251
concat = $a362:  2816 , 23781#
conint = $87f7: 13655 , 15496 , 15724 , 15921 , 16052 , 16150 , 16257 , 16369 , 16440
                16466 , 16490 , 16608 , 17102 , 17658#, 21878
cont   = $5a60:  2747 ,  6856#
contlp = $5a91:  6899#,  6901
contrt = $5a9a:  6856 ,  6859 ,  6906#
contz3 = $5a70:  6866 ,  6873#
conupk = $8ab4: 17718 , 17742 , 18191 , 18319#, 18467 , 25538
conv   = $95a0: 20830 , 20871#
conv1  = $95a5: 20874#, 20876
conv2  = $95b4: 20884#, 20888
conv4  = $95be: 20885 , 20890#
coout  = $90eb:  5638 , 19792#
copen  = $90d8: 19771#, 19946 , 23202 , 23362 , 23838 , 24828
copflt = $53fa:  5319 ,  5341#
coplin = $7663: 13270 , 13466#
copstr = $5404:  5312 ,  5349#
copy   = $5494:  5442 ,  5446 ,  5477#,  9262
copyc  = $54b2:  5450 ,  5496#
cordsb = $9f08: 22688 , 22756 , 22789#
cos    = $9409:  2850 , 20532#, 25529
cosc   = $9481: 20605 , 20612#
cosval = $114c:   529#,  8893 ,  8922 , 21832
count  =   $0d:    96#,  2313 ,  2358 ,  2410$,  2439$,  2460 ,  2461$,  3859$,  3875
                3987 ,  4089 ,  6463$,  6483 ,  6504$,  6506 , 15078$, 15081 , 15171
                15205 , 15244$, 15311$, 15372$, 15469
cpef   = $9591: 20852 , 20859#
cplot  = $928d:  5722 ,  5738 ,  9817 , 10494 , 16882 , 20150#
cprimm = $9281:  3423 ,  3700 ,  3803 ,  6158 ,  6223 , 19159 , 19889 , 20138#, 24876
cprts  = $5df8:  7709 ,  7713#
cpydlp = $54eb:  5544#,  5548
cpyds  = $79d2: 14337#, 14343
cpydsc = $54e9:  5527 ,  5543#
crcseg = $1220:   622#,  9592$,  9599
crdo   = $5598:  3788 ,  4557 ,  4597 ,  5656 ,  5706#,  6663 ,  6671 ,  6927 , 20909
crdsb1 = $9f14: 22791 , 22794 , 22801#
creads = $9251:  5931 , 19882 , 19896 , 20090#, 23261 , 23267 , 23286 , 23566
```

```
crfin  = $55a6:  5710 ,  5715#,  5801
crne10 = $4365:  2279 ,  2290#
crne20 = $4374:  2293 ,  2304#
crne30 = $4384:  2310 ,  2313#
crnesc = $4356:  2262 ,  2276#
crtskp = $5607:  5808 ,  5819#
crun05 = $4313:  2213#,  2256 ,  2327
crun10 = $4319:  2219#,  2220 ,  2239 ,  2252 ,  2270 ,  2307 ,  2321 ,  2382
crun20 = $431c:  2214 ,  2220#
crun25 = $4326:  2229 ,  2235#
crun30 = $4336:  2242 ,  2248#
crun40 = $4345:  2249 ,  2261#
crun50 = $4349:  2264#,  2271
crun75 = $4386:  2245 ,  2314#
crun80 = $439b:  2318 ,  2332#
crun90 = $43a1:  2236 ,  2267 ,  2338#
crun94 = $43b0:  2285 ,  2299 ,  2358#
crun95 = $43b2:  2230 ,  2363#
crun96 = $43be:  2371 ,  2374#
crunch = $430a:  2199#,  3850 ,  3858 , 25554
cscol  = $9556: 20811 , 20821#
csetnm = $925d: 19969 , 20030 , 20070 , 20102#, 24612 , 24827
cstbnk = $9287: 19975 , 20028 , 20144#, 23194 , 23360 , 23427 , 23457 , 23498 , 23549
                23835
cstlfs = $9257: 19973 , 19985 , 19992 , 20039 , 20051 , 20059 , 20096#, 23199 , 24617
                24824
cstop  = $9293:  3369 ,  3384 , 20157#, 23301
ctoklp = $5c29:  7285#,  7289
curbnk = $03d5:   475#,  1759$,  6423 , 10764$, 10856 , 12718 , 15694 , 15718 , 23922
                 25427
curlin =   $3b:   152#,  3207$,  3211$,  3411 ,  3412 ,  3737 ,  3816 ,  3838$,
                 4980$,  5139$,  5143$,  6386$,  6390$,  6728 ,  6731 ,  6802 ,
                 6806 ,  6881$,  6882$,  7521$,  7524$,  7763 ,  7767 ,  8089$,
                 8093$,  8122$,  8166 ,  8289$,  8293$,  8345$,  8346$,  8383$,
                19165 , 19166 , 24955$
curprt = $8e2e:  3257 , 19165#
czloop = $4f67:  4170#,  4172
d1cra  = $dc0e:  1538#
d1crb  = $dc0f:  1575#
d1ddra = $dc02:  1445#
d1ddrb = $dc03:  1452#
d1icr  = $dc0d:  1509#
d1pra  = $dc00:  1426#, 16065 , 16072$, 16077 , 16083 , 16089$, 16179 , 16182$,
                16226$, 16230 , 16239$
d1sdr  = $dc0c:  1503#
d1t1h  = $dc05:  1460#, 16765
d1t1l  = $dc04:  1459#, 16762
d1t2h  = $dc07:  1465#, 16759
d1t2l  = $dc06:  1464#, 16756
d8502  =   $00:   773#
data   = $528f:  2326 ,  2724 ,  2822 ,  4982 ,  4987#,  5073 ,  5941 ,  8106 ,  8307
                16965
datan  = $52a2:  4987 ,  5010#,  5950 ,  6232 ,  7746
datbk  = $56fe:  6029 ,  6052#
datbk1 = $5702:  5997 ,  6058#,  6265
datlin =   $41:   155#,  6250$,  6256$
datlop = $57ca:  6038 ,  6232#,  6263
```

```
datptr =   $43:   156#,  5966 ,  5967 ,  6208$,  6209$,  6982$,  6983$, 22960 ,
                22962$, 23099 , 23101$
dbank  = $a59e: 24019 , 24306 , 24332#
dbank1 = $a442: 23971 , 24019#
dcat0  = $a08a: 23159 , 23166#
dcat1  = $a099: 23177 , 23187#
dcat10 = $a114: 23209 , 23262 , 23268 , 23302 , 23315#
dcat2  = $a09a: 23171 , 23189#
dcat21 = $a0bd: 23203 , 23217#
dcat22 = $a0c7: 23256#, 23305
dcat23 = $a0ca: 23258#, 23271
dcat4  = $a0f3: 23283#, 23293
dcat5  = $a105: 23290 , 23298#
dcat9  = $a113: 23287 , 23310#
dclall = $a183: 23401 , 23409#, 23729 , 23822
dclear = $a322:  2818 , 23724#
dclose = $a16f:  2812 , 23392#
dcml   = $8076:  2869 , 15608#
dcopy  = $a346:  2788 , 23755#
dcplop = $8028: 15536#, 15544
dcpy2  = $a355: 23758 , 23769#
dcpy4  = $a35c: 23764 , 23774#
dec010 = $8083: 15620#, 15627 , 15668
dec030 = $80a9: 15638 , 15653#
dec040 = $80b1: 15662#, 15667
dec050 = $80bb: 15621 , 15673#
decbad = $80c2: 15632 , 15635 , 15641 , 15644 , 15680#
decblt = $7ca3: 14979 , 15015#
deccnt =   $5f:   203#, 14295$, 15427$, 15461$, 19007 , 19039$, 19223$, 19246$,
                19250$, 19258 , 19274$, 19352$
deccur = $7db5: 15267 , 15276#
decy   = $9717: 21107#, 21119
def    = $84fa:  2743 , 16937#
def040 = $7497: 13033#, 13483 , 13484 , 13485 , 13486
def044 = $74e0: 13072 , 13091#
def046 = $74ea: 13088 , 13093 , 13099#
def050 = $74f4: 13112#, 13487
def051 = $74f6: 13113#, 13116
def060 = $74fe: 13121#, 13488
def070 = $7506: 13132#, 13491
def080 = $751c: 13151#, 13490
def090 = $751f: 13160#, 13489
def100 = $7530: 13176#, 13496
def102 = $7534: 13178#, 13180
def110 = $753f: 13105 , 13188#, 13493
def120 = $7542: 13194#, 13492
def121 = $754a: 13200 , 13204#
def122 = $7559: 13206 , 13217#
def123 = $7561: 13223 , 13227#
def130 = $7567: 13238#, 13263 , 13495
def140 = $7564: 13232#, 13494
def150 = $7576: 13252#, 13498
def155 = $757e: 13212 , 13244 , 13247 , 13256#
def160 = $7581: 13209 , 13261#, 13499
defbeg = $73c4: 12839#, 13127
defcl1 = $76dc: 13530#, 13540
defcl2 = $76e3: 13535#, 13537
defclr = $76d4: 12834 , 13182 , 13526#
```

```
defcor = $9e5e: 22659 , 22668#
defcp1 = $758c: 13270#, 13275
defcp2 = $7597: 13271 , 13280#
defcp3 = $759a: 13285#, 13286 , 13295
defcp4 = $75b4: 13303 , 13305#
defcp5 = $75b8: 13308#, 13311
defcp6 = $75c2: 13289 , 13315#
defcp7 = $75c5: 13317#, 13319
defcpy = $7588: 13268#, 13500
deffin = $85a0: 16966 , 17069#
defflp = $85a5: 17072#, 17076
defjmp = $7691: 13022 , 13025 , 13483#
defmod = $12fa:   693#, 12895$, 13068 , 13101 , 13132 , 13134$, 13196 , 13219 , 13252
                13254$, 13346 , 13385 , 13429 , 13448
defpnt =   $50:   167#,  2121 , 16977$, 16978$, 16987 , 16990 , 16997$, 17000$,
                17047$, 17050$, 17073$, 19539 , 19540 , 19548 , 19549 , 19561 ,
                19562 , 22910$, 22914$, 22923$, 22926$
defst1 = $859a: 17053 , 17060#
defstf = $8561: 17014#, 17018
del1   = $a4f7: 24008 , 24091 , 24171#
del2   = $a568: 24248 , 24254 , 24260 , 24277#
del300 = $5eb4:  7870 ,  7879 ,  7891#
del500 = $5ed2:  7916#,  7919 ,  7928
delete = $5e87:  2791 ,  7856#
delim1 = $a4fb: 24002 , 24113 , 24134 , 24177#
devn   = $96af: 21032 , 21038#
devn2  = $96b2: 21035 , 21039#
dfloop = $745d: 12973#, 12974
didchk = $0122:   416#, 24100 , 24110$
dim    = $587b:  2727 ,  6402#
dim3   = $5878:  6397#,  6408
dimflg =   $0e:   100#, 14541$, 15025 , 15095$, 15158 , 15214 , 15296
dimrts = $7e3d: 15301 , 15408#
diris  = $4be1:  3407 ,  3420#
disend = $115c:   557#, 21870
div10  = $8b38: 18449#, 19013 , 19249
divide = $8b63: 18491#, 18528
divnrm = $8bb8: 18513 , 18557#
divp10 = $9d34: 22345 , 22349#
divp20 = $9d48: 22336 , 22351 , 22354 , 22370#
divpos = $9d24: 22068 , 22264 , 22334#
divsub = $8b97: 18517 , 18532#
dlimx1 = $a4c4: 24131#, 24148
dload  = $a1a7:  2784 ,  6919 , 23440#
dloop  = $9ba5: 21996#, 22030
dloop1 = $7469: 12983#, 13001
dloop2 = $747b: 12984 , 13000#
dloop3 = $7480: 13006#, 13010
dma    = $df00:  1627#, 25402$, 25404$, 25408$, 25410$, 25414$, 25416$, 25423$
dmoerr = $84f5: 16924 , 16931#
dnote  = $1233:   632#, 11559$, 11713 , 11768$
dntcpy = $5476:  5431 ,  5437 ,  5441 ,  5448#
do     = $5fe0:  2779 ,  8158#,  8385
do10   = $600b:  8176 ,  8197#
doagin = $5764:  6144 ,  6158#
doang1 = $9ead: 22726#, 22745
doang2 = $9ec0: 22734 , 22737#
docl10 = $9c86: 22150 , 22161#
```

```
docl20 = $9c8d: 22154 , 22167#
docl30 = $9c9a: 22162 , 22180#
docl40 = $9cac: 22180 , 22198#
docmp  = $4d1e:  3667 ,  3670 ,  3685 ,  3688#
docolr = $9c70:  9957 , 22071 , 22143#
docor1 = $9eee: 22766#, 22781
docor2 = $9eff: 22769 , 22779#
docord = $9ed3: 22692 , 22753#
docstr = $77e5: 13772 , 13780#
doesc1 = $4c68:  3526 ,  3536#
doescf = $4c54:  3439 ,  3522#
doffh  = $a4ca: 24139#, 24205
doffl  = $a4b4: 23981 , 24118#
dohex  = $8174: 15847 , 15851#
dohex1 = $817c: 15853 , 15856#
dohome = $7452: 12961#, 13183 , 13497
dolop1 = $5fe2:  8163#,  8170
dolr   = $0125:   427#, 21214$, 21292 , 21389 , 21393$, 21490 , 21494$
domin  = $7971: 14130 , 14244#
don0   = $6005:  8191#,  8200 ,  8264
done   = $a3ea: 23931#, 24180
done1  = $a500: 24180#, 24281
dop2   = $a143: 23337 , 23340 , 23354#
dopen  = $a11d:  2810 , 23324#
dopre1 = $7871: 13931 , 13975#
doprec = $784a: 13930#, 13970
dorel  = $4cb6:  2914 ,  3612#
dores  =   $11:   114#,  4605$,  4626 ,  4631$,  4660 , 14452$, 14456 , 14457$, 14465
                20169$, 20233 , 20244$
dorts  = $4bf6:  3401 ,  3433#
dos01  = $a3ce: 23908#, 23910
dos02  = $a3d6: 23915#, 23918
dos5   = $a41f: 23977 , 23984#
dosbnk = $011f:   411#, 12727 , 23496 , 23547 , 23923$, 24339$
doscal = $9d5a: 22383 , 22391#
dosdid = $0120:   413#, 23600 , 24104$, 24107$, 24647 , 24651
dosds1 = $0112:   380#, 12745 , 24087$, 24575
dosds2 = $0114:   386#, 24088$, 24245$, 24584
dosf1l = $0111:   377#, 24155$, 24163 , 24687 , 24698
dosf2a = $0115:   389#, 24269$, 24270$, 24705 , 24708
dosf2l = $0113:   383#, 24268$, 24711 , 24723
dosfa  = $011c:   396#,  1737$, 12747 , 23197 , 23409 , 23698$, 24324$, 24615 , 24814
                24820$
dosla  = $011b:   393#, 23403 , 23677$, 23692 , 23701$, 23843 , 24032$, 24614
dosofh = $0119:   391#, 23482 , 23487 , 23508 , 23509 , 24144$, 24145$
dosofl = $0117:   390#, 23483 , 23488 , 23500 , 23501 , 23530 , 23531 , 23552 , 23553
                23916$, 24123$, 24124$
dospar = $a3bf: 23155 , 23584 , 23628 , 23724 , 23734 , 23755 , 23781 , 23872#
dosprs = $a3c1: 23325 , 23346 , 23393 , 23417 , 23444 , 23793 , 23805 , 23886#
dosprx = $a3c3: 12704 , 23466 , 23523 , 23896#
dosrcl = $011e:   406#, 23690$, 24069$, 24566 , 24657 , 24669$
dossa  = $011d:   399#, 23379$, 23449$, 23541$, 23704$, 24616
dosstr = $1100:   509#, 24594$, 24610 , 24611 , 24648$, 24678$, 24693$, 24718$
dostbl = $a3b8: 23860#, 23915
dotwo  = $9d6b:  9649 , 22410#, 22737
dotwo2 = $9d67:  9656 , 22404#
doverr = $8b33: 18443#, 18472
doyes  = $6012:  8173 ,  8186 ,  8205#
```

```
dptflg =   $61:   199#, 19001$, 19002 , 19036
draw   = $6797:  2773 ,  9681#
draw10 = $67b0:  9688 ,  9695 ,  9700#,  9722 ,  9729
draw20 = $67bc:  9702 ,  9705 ,  9710#
draw30 = $67ce:  9717 ,  9727#
drawln = $9b30:  8834 ,  8876 ,  9612 ,  9618 ,  9721 , 21913#, 25546
drv1   = $a47f: 23965 , 24080#
drv2   = $a53a: 24221 , 24238#
drw01  = $9b34: 21916#, 21952
drw02  = $9b49: 21921 , 21930#
drw03  = $9b51: 21931 , 21935#
drw04  = $9b54: 21925 , 21933 , 21936#
drw05  = $9bc1: 22003 , 22008 , 22013#
drw06  = $9bcf: 22015 , 22019#
drwin1 = $9bed: 22037#, 22043
drwinc = $9bea: 22017 , 22029 , 22035#
dsave  = $a18c:  2783 , 23416#
dscfre = $7799: 13581 , 13651 , 13681#
dsch10 = $924e: 20080 , 20083#
dschk  = $9243: 19796 , 19813 , 19843 , 19873 , 19947 , 19962 , 20075#, 23557
dscpnt =   $52:   171#,  2133 ,  5481 ,  5482 ,  5496$,  5497$,  5544 , 17153 , 17154
                17244$, 17247$, 17300$, 17301$, 17452 , 17453
dsctmp =   $63:   208#,  3635$,  3636$,  3637$,  5493 ,  5494 , 17113$, 17307$,
                17308$, 17309$, 17326$, 17327$, 17345$, 17392 , 17395 , 17398
dsdesc =   $7a:   265#,  4858$,  5457 ,  5464 ,  5467 ,  5583 ,  5586 , 14322 , 14338
                14354 , 14445 , 14462 , 23612 , 23648 , 24790 , 24796$, 24799$,
                24800$, 24807 , 24808$, 24811 , 24812$, 24842$, 24851$, 24904 ,
                24913$, 24917$, 24921$
dskx1  = $5458:  5424#,  5465 ,  5468
dskx2  = $545a:  5430#,  5458
dstmov = $9df9:  9087 , 22546 , 22553#
dstpos = $9df2:  8594 ,  8737 ,  8994 ,  9263 ,  9345 ,  9578 ,  9727 , 10102 ,
                22545#
dummy  =   $00:    68#,  2070 ,  2081
dvarts = $7c33: 14870 , 14900#
dverfy = $a1a4:  2817 , 23434#
eadj   = $973f: 21117 , 21137#
eado   = $973d: 21097 , 21136#, 21280
eatem  = $7ad4: 14576#, 14577 , 14580
echks  = $a795: 24791 , 24814#
eex2   = $9558: 20822#, 20895
efnvec = $02fc:   455#,  2012$,  2015$,  3560
efo    = $99a0: 21459 , 21514 , 21539#
else   = $5391:  2757 ,  5227#
elsrem = $53a0:  5228 ,  5236 ,  5239#
end    = $4bcd:  2721 ,  3400#
endchr =   $0a:    87#,  3592$,  3599 ,  5020$,  5022 ,  5026$,  5031 ,  6094$,  7981
                17318$, 17337
endfd  = $0136:   444#, 20777$, 21356 , 21427 , 21529$, 21539$, 21541$
endrel = $7828: 13844 , 13847 , 13878#
endv10 = $57a9:  6204 ,  6213#
endvar = $579c:  6187 ,  6197#
enr    = $0124:   426#, 20946$, 20981 , 21090 , 21231 , 21249$, 21282 , 21370
envlop = $70c1:  2807 , 12067#
eoa    = $9631: 20929 , 20964#
eread  = $a79f: 24815 , 24822#
erexit = $90d0: 12759 , 19756#, 19779 , 19786 , 19803 , 19814 , 19844 , 19965 , 23563
                23855
```

```
erexiy = $90d5: 19757 , 19766#
ero    = $993e: 21474#, 21504 , 21507 , 21526
errbad = $a7d5: 24832 , 24861#
errchl = $a778: 14357 , 23606 , 23638 , 24790#
errd   = $80f6:  2871 , 15724#
errd1  = $8106: 15738#, 15749
errd2  = $8107: 15743#, 15748
errd3  = $8112: 15745 , 15754#
errd4  = $811d: 15764#, 15767 , 15779
errdir = $84d9:  5837 ,  5921 ,  8026 ,  8047 , 16893#, 16938
errend = $a7c8: 24840 , 24850#, 24863
rrerr  = $8136: 15728 , 15786#
errf   = $96d6: 20997 , 21003 , 21042 , 21053 , 21062#
errfin = $4daf:  3427 ,  3816#
errgo3 = $7d2a: 15151#, 15163
errguf = $84e0: 16904#, 17009
errild = $a5ea: 24319 , 24322 , 24399#
errind = $84f0:  6634 ,  7008 ,  7856 , 16923#
errisd = $4d7c:  3732 ,  3750 ,  3771#
errlen = $a5ed:  4228 ,  9190 , 17446 , 24384 , 24405#
errlin = $1209:   609#,  3738$,  4862$,  4863$,  6653 ,  6654 ,  8051 ,  8121 ,
                 8136$,  8137$, 14484 , 14485
errmfn = $a5e7: 24360 , 24393#
errnlg = $5b98:  7141 ,  7155#
errnum = $1208:   608#,  3729$,  4864$,  6649 ,  8134$, 14493
erro1  = $4d4e:  3737#,  3744
error  = $4d3c:  2926 ,  3390 ,  3719#,  4957 ,  5200 ,  6153 ,  6313 ,  6778 ,  6870
                 7156 ,  7526 ,  8153 ,  8355 , 13792 , 14239 , 15151 , 16905 , 16932
                17389 , 18001 , 18444 , 19766 , 19917 , 23144 , 23212 , 23387 , 23620
                23719 , 24406 , 24868
error2 = $4d8e:  3781 ,  3788#
errtab = $484b:  2932#,  3060 ,  3063
errtmf = $a16a: 23373 , 23386#
errtxt = $120e:   612#,  3741$,  6685 ,  6688 ,  8124 , 22994 , 22995$
erst   = $966a: 20973 , 20997#
erst10 = $4a8d:  3069#,  3084
erst20 = $4a90:  3075#,  3083
erst30 = $4a99:  3079 ,  3082#
erst40 = $4a9e:  3070 ,  3089#
erstup = $4a82:  3057#,  3774 , 15730
esccmd = $4609:  2276 ,  2277 ,  2648#,  4737 ,  4738
escfn  = $46c9:  2290 ,  2291 ,  2696#,  4761 ,  4762
escp10 = $51d3:  4800 ,  4806#
estdsp = $478e:  2799#,  3340
et2    = $9774: 20982 , 21171#
et3    = $976f: 21160 , 21169#
ete    = $965d: 20979 , 20990#
etof   = $0130:   438#, 21018$, 21140 , 21159
eval   = $78d7: 13826 , 14081#, 17428 , 25557
eval0  = $78de: 14089#, 14133
eval1  = $78e3: 14095#, 14127
eval2  = $78e8: 14090 , 14101#
eval3  = $7928: 14136 , 14161#
eval4  = $7942: 14162 , 14186#
eval5  = $78f0: 14102 , 14109#
eval6  = $7949: 14187 , 14194#
exchqt = $52ad:  5022#,  5037
execln = $4dcd:  3841#, 25563
```

```
exit   = $6039:  2781 ,  8237#
exp    = $9033:  2849 , 19531 , 19565 , 19632#, 25528
exp1   = $9051: 19647 , 19650#
expco0 = $902e: 19627#
expco1 = $9029: 19626#
expco2 = $9024: 19621#
expco3 = $901f: 19616#
expco4 = $901a: 19611#
expco5 = $9015: 19606#
expco6 = $9010: 19601#
expco7 = $900b: 19596#
expcon = $900a: 19591#, 19679 , 19680
expo   = $9964: 21488 , 21499#
expsgn =   $62:   201#,  7826 , 18982$, 18990 , 19074
fa     =   $ba:   321#, 20078
facexp =   $63:   213#,   214#,   214#,   216#,   218#,   220#,   222#,   224#,  3647
                 3653 ,  5054 ,  5367$,  7788 ,  8185 ,  8199 ,  8324 ,  8651$,  8680
                 8682$, 10204 , 14028 , 14075 , 14507$, 16803 , 16807$, 16854 , 17697
                17730 , 17772 , 17778$, 17789 , 17806 , 17878$, 17922 , 17927$,
                17937$, 18123 , 18128$, 18313 , 18355 , 18375 , 18386$, 18430$,
                18478 , 18479$, 18482$, 18611$, 18656 , 18675$, 18693 , 18706 ,
                18723 , 18773$, 18803 , 18845 , 18860 , 18886 , 18899$, 19063 ,
                19201 , 19574 , 19645 , 19665, 19667$, 20674
facho  =   $64:   215#,   216#,  3676 ,  4124$,  7215 ,  7559$,  7585$,  7621 ,
                 7623$,  7781 ,  7782$,  8652$,  8691 ,  8693$,  9194$,  9232$,
                 9244 , 14025 , 14413$, 14516$, 15773$, 15857$, 16757$, 16789 ,
                16792$, 16913$, 17834$, 17850 , 17856$, 17902 , 17904$, 17916$,
                17942$, 17959 , 17961$, 17994$, 18223 , 18492 , 18547 , 18571$,
                18607$, 18652 , 18749$, 18759 , 18809 , 18874$, 18875 , 18876$,
                18909$, 19171$, 19316 , 19318$, 20886$
faclo  =   $67:   221#,   222#,  3590 ,  3597 ,  5266$,  5333 ,  5405 ,  5439 ,  5449
                14016 , 14172 , 14410$, 14528$, 15074 , 15331$, 15369 , 16766$, 16788
                16791$, 16835 , 17663 , 17704 , 17822$, 17861 , 17865$, 17890 ,
                17892$, 17913$, 17945$, 17971 , 17973$, 17985$, 18214 , 18504 ,
                18535 , 18580$, 18592$, 18638 , 18767$, 18830 , 18901 , 18912$,
                19303 , 19306$, 20788 , 21593$, 21605
facmo  =   $66:   219#,   220#,  2151 ,  3586 ,  3603 ,  5327 ,  5403 ,  5444 ,  5448
                14019 , 14176 , 14408$, 14524$, 15071 , 15325$, 15364 , 16760$, 16795
                16798$, 16834 , 17660 , 17703 , 17826$, 17858 , 17862$, 17894 ,
                17896$, 17914$, 17944$, 17967 , 17969$, 17988$, 18217 , 18500 ,
                18539 , 18577$, 18596$, 18642 , 18768$, 18821 , 18911$, 19308 ,
                19310$, 20785
facmoh =   $65:   217#,   218#,  4115$,  5404 ,  7219 ,  7556$,  7582$,  7616 ,
                 7619$,  9247 , 14022 , 14409$, 14520$, 16763$, 16794 , 16797$,
                16915$, 17830$, 17855 , 17859$, 17898 , 17900$, 17915$, 17943$,
                17963 , 17965$, 17991$, 18220 , 18496 , 18543 , 18574$, 18600$,
                18646 , 18752$, 18815 , 18910$, 19172$, 19312 , 19314$
facov  =   $71:   234#, 14266$, 14509$, 15534$, 15536 , 15542$, 16804$, 17754 ,
                17793$, 17799 , 17818$, 17864 , 17867$, 17888$, 17912$, 17946$,
                17975 , 17977$
                17979$, 18012$, 18036 , 18211 , 18271$, 18563$, 18613$, 18659$,
                18680$, 18699$, 18709$, 18775$, 18827 , 18891$, 19636 , 19674$
facsgn =   $68:   223#,   224#,  3655$,  3662 ,  5368$,  6330$,  7779 , 10201 , 13996
                14069 , 14513$, 16801$, 16848 , 17691 , 17723 , 17725$, 17781$,
                17880$, 17952 , 17954$, 18302 , 18341 , 18394$, 18399 , 18604$,
                18650 , 18671$, 18729 , 18776$, 18783$, 18800 , 18836 , 18850 ,
                18872 , 18893 , 18894$, 19060 , 19189 , 19195$, 19577 , 19579$,
                20555 , 20561 , 20602$, 20668
```

```
fadd   = $8845:  6345 , 17742#, 25516
fadd1  = $8876: 17790 , 17795#
fadd2  = $88dd: 17804 , 17887#
fadd4  = $8882: 17737 , 17773 , 17803#
fadd5  = $8840: 17736#, 17796
fadda  = $8872: 17774 , 17792#
faddb  = $884d: 17747 , 17754#
faddc  = $8855: 17763#, 18429
fadde  = $8859: 17764 , 17771#
faddh  = $8a0e: 18166#, 19253 , 20560
faddt  = $8848:  2887 , 17731 , 17747#, 18174 , 19064 , 25517
fadflt = $88b1: 17839#, 18778 , 18904
fast   = $77b3:  2834 , 13705#
fbuffr = $0100:   371#,  4134 ,  7384 ,  7409 ,  7439 , 17095 , 17096 , 19193$,
                 19285$, 19292$, 19350$, 19357$, 19376 , 19400$, 19403$, 19413$,
                 19416$, 19419$, 19425$, 19431$, 19433 , 19434 , 20875 , 20884 ,
                 20931 , 20945$, 20956 , 20983$, 21129$, 21153 , 21154$, 21174$,
                 21177$, 21182 , 21183$, 21240 , 21257$, 21274 , 21353 , 21374 ;
                 23908$
fbufpt =   $72:   242#,  5373$,  5379$,  5383 ,  5395 , 14152 , 14153 , 14292$,
                 15184$, 15203$, 15239$, 15240$, 15265$, 15266 , 15277$, 15314$,
                 15319$, 15356 , 15357 , 15370$, 15375$, 15444$, 15445$, 15576$,
                 15580$, 15582 , 15583 , 15588$, 15601$, 15614$, 15615$, 15663$,
                 15664$, 15673 , 15674 , 19196$, 19282 , 19294$, 19345 , 19359$, 19374
fcer   = $a31a: 23675 , 23685 , 23688 , 23713#
fcerr  = $7d28:  1697 ,  1698 ,  5417 ,  6629 ,  7060 ,  8009 ,  8403 ,  8602 ,  8814
                 9317 ,  9589 ,  9769 , 10113 , 10213 , 10354 , 10451 , 10746 , 10770
                10960 , 11554 , 11788 , 11996 , 12062 , 12076 , 12272 , 12308 , 12328
                12365 , 12380 , 12691 , 13676 , 15147#, 15504 , 15680 , 15786 , 15955
                16128 , 16251 , 16424 , 16460 , 16485 , 16563 , 16650 , 16866 , 17292
                17713 , 18120 , 21610 , 21885 , 22643 , 23516 , 23713 , 24095 , 24299
                25431
fcerr2 = $5450:  5364 ,  5417#
fcerr3 = $7796: 13659 , 13676#
fchi   =   $16:  1050#
fclow  =   $15:  1045#, 11460$
fcomp  = $8c87:  3622 ,  6367 , 16863 , 18789#, 19230 , 19240 , 19550 , 25536
fcompc = $8cbe: 18804 , 18810 , 18816 , 18822 , 18836#
fcompd = $8cc4: 18837 , 18840#
fcomps = $8c5d: 18734#, 18840
fconc  =   $0d:    97#,  3582$,  3587 ,  3591 ,  3598 ,  3600 ,  3604 ,  3606
fcsign = $8c5b: 18729#, 18801
fct    = $1141:   524#,  8961$,  8966$, 21939$, 21945$, 21993$, 21994$, 22014 , 22020
                22021 , 22022$, 22024 , 22025 , 22026$
fdcend = $8f9f: 19368 , 19486#
fdiv   = $8b49: 18467#, 20609 , 25520
fdivf  = $8b41: 18459#, 20544
fdivt  = $8b4c:  2896 , 10151 , 10172 , 18186 , 18462 , 18472#, 25521
fesp   = $012f:   437#, 20975 , 21512$
fetch  = $aa24:  2830 , 25391#
fform  = $95e7: 20892 , 20915#
ffoun  = $991c: 21433 , 21451#
fgbg   = $03e2:   483#, 10393$, 10609 , 22152 , 22184
fgmc1  = $03e3:   484#, 10399$, 10614 , 22166
fhalf  = $8f76: 18166 , 18167 , 19440#, 19524 , 19525
fiderr = $575f:  6135 ,  6149#
filflg = $116c:   572#,  1735$,  8817$,  8845$, 22050
filt10 = $7049: 11936#, 11939
```

```
filt20 = $7070: 11945 , 11971#
filt30 = $7075: 11982#, 12013
filt40 = $7083: 11991 , 11992 , 11999#
filt50 = $708e: 12001 , 12007#
filt60 = $7091: 11986 , 12012#
filt70 = $70b2: 12018 , 12051#
filt80 = $70b4: 12052#, 12055
filter = $7046:  2800 , 11933#
fin    = $8d22:  6112 , 14096 , 15595 , 18926#
fin1   = $8d41: 18947 , 18955#
finbnk = $03da:   478#, 18926$, 19096 , 19115
finc   = $8d3c: 18943 , 18952#, 19003 , 19047
find10 = $5d2e:  7514 ,  7529#
find20 = $5d4d:  7545 ,  7555#
find30 = $5d55:  7546 ,  7550 ,  7564#
find40 = $5d5e:  7533 ,  7572#
find50 = $5d63:  7539 ,  7575#
findg1 = $8da3: 19037 , 19040#
findgq = $8d3f: 18937 , 18953#
findig = $8d9c: 18953 , 19034#
findit = $5d1c:  7513#,  7576
findiv = $8d84: 19013#, 19015
findp  = $8d73: 18956 , 19001#
fine   = $8d79: 18959 , 18991 , 19005#
fine1  = $8d7b: 18996 , 19006#
finec  = $8d62: 18973 , 18976 , 18987#, 19110
finec1 = $8d60: 18967 , 18970 , 18982#
finec2 = $8d67: 18977 , 18990#
finedg = $8dc3: 18988 , 19069#
fingo  = $4c3f:  3496 ,  3505#,  3543
fininl = $558b:  4217 ,  5695#
finlog = $8db0:  5421 , 18151 , 19045 , 19052#
finml6 = $8b22:  5393 , 18426#
finmul = $8d8d: 19011 , 19019#, 19021
finpt1 = $7c61: 14938 , 14944#
finptr = $7c57: 14665 , 14934#
finqng = $8d94: 19010 , 19016 , 19023#
finre2 = $7862: 13951 , 13953#
finrel = $7858: 13879 , 13942#
finzlp = $8d29: 18932#, 18935
fixlnk = $4f82:  4198#, 22968
flag   = $0126:   428#, 21212$, 21304 , 21378$
flinrt = $509e:  4429 ,  4469#
flnrts = $509f:  4438 ,  4452 ,  4453 ,  4470#
fload  = $5235:  4898#, 19939
float  = $8c68:  3695 , 14429 , 14466 , 14494 , 18749#, 19057
floatb = $8c7b: 14402 , 18773#
floatc = $8c75:  4128 ,  7566 , 16876 , 18766#, 19176 , 25513
floats = $8c70: 14183 , 18759#
flpint = $849f:  1703 ,  1704 , 16833#
flterr = $70be: 11954 , 12023 , 12062#
flters = $1271:   644#,  1863$, 11430 , 11445$, 11447 , 11458 , 11525 , 11535$, 11936
                 12053$, 12342 , 12348$
fltflg = $1238:   634#, 11977$, 12000 , 12005 , 12012$
fltsav = $1234:   633#, 11937$, 11956$, 11957$, 11960$, 11963$, 11966$, 11999 ,
                 12007$, 12035 , 12049$, 12052
fltvol =   $18:  1066#,  1865$, 11541$, 12359$
fmapt1 = $7e7d: 15475 , 15478#
```

```
fmaptr = $7e71: 15165 , 15180 , 15469#
fmult  = $8a24: 18191#, 19563 , 19699 , 19704 , 25518
fmult1 = $8a2c: 18196 , 18203#
fmultt = $8a27:  2893 , 18161 , 18196#, 25519
fnadr  =   $bb:   322#
fnbank =   $c7:   324#
fnchg1 = $8dfd: 19116 , 19123#, 19133
fnchg2 = $8e03: 15592 , 19124 , 19127#
fnchg3 = $8e16: 19130 , 19139#
fnchgt = $8df5: 18952 , 18963 , 18987 , 19115#
fnd010 = $6047:  8192 ,  8240 ,  8250#
fnd40  = $605b:  8256 ,  8269#
fnd50  = $6060:  8250 ,  8275#
fnd60  = $6087:  8253 ,  8307#
fndb10 = $5325:  5120#,  5163
fndb20 = $5329:  5126#,  5173
fndb30 = $534c:  5118 ,  5157#
fndb40 = $5357:  5158 ,  5169#
fndb50 = $5361:  5170 ,  5178#
fndbnd = $5320:  5068 ,  5117#,  5121 ,  5152 ,  5154 ,  5164 ,  5179 ,  5188 ,  5190
                 5191 ,  5238
fndbrt = $5376:  5183 ,  5194#
fndend = $6044:  8249#,  8259 ,  8263 ,  8270 ,  8276 ,  8302 ,  8304
fnderr = $60aa:  8279 ,  8283 ,  8342#
fndesc = $5c4d:  7321#,  7325
fndlin = $5064:  3861 ,  4417#,  6659 ,  6957 ,  7078 ,  7094 ,  7869 ,  7967 , 25555
fndln3 = $5085:  4439 ,  4445#
fndlnc = $5068:  4423#,  4464 ,  6756
fndoev = $853b: 14191 , 16985#
fndpnt =   $3f:   154#,  2127 ,  4239 ,  4243 ,  4256 ,  4267 ,  4272 ,  4279 ,  4294
                 4295$,  4297$,  4384$,  4387$,  4394 ,  4397 ,  4968 ,  4972 ,  4976
                 4979 ,  6318 ,  6322 ,  6329 ,  6333 ,  6339 ,  6358 ,  6362 ,  6371
                 6377 ,  6381 ,  6385 ,  6389 ,  7183$,  7184$,  7393$,  7395$,  7401
                 7404 ,  7420 ,  7422$,  7424$,  8371 ,  8375 ,  8379 ,  8382 ,
                20796$, 23002 , 23006 , 23016 , 23036 , 23037$, 23039$, 23108 ,
                23121$, 23129$
fndsca = $a157: 23328 , 23349 , 23370#
fnedg1 = $8d65: 18964 , 18988#
fnlen  =   $b7:   319#
fnoerr = $a31d: 23695 , 23718#
fone   = $899c:  7803 ,  7804 , 18064#, 18138 , 18139 , 20680 , 20681
for    = $5df9:  2722 ,  7724#
for010 = $5e0f:  7733 ,  7745#
for020 = $5e45:  7788#,  7799
for030 = $5e61:  7809 ,  7815#
for040 = $5e6d:  7825#,  7829
foregn = $4c74:  3537 ,  3540 ,  3548#
forgnd =   $86:   292#,  1764$,  9860 ,  9889 ,  9898$,  9950$, 10282$, 10379 , 15971
fouldy = $8f29: 19369 , 19374#
foun1  = $9917: 21440 , 21448#
fout   = $8e42:  4129 ,  5686 ,  7567 , 19183#, 20871 , 25510
fout1  = $8e4c: 19190 , 19193#
fout10 = $8e5d: 19202 , 19209#
fout11 = $8f2b: 19376#, 19380
fout12 = $8f38: 19383 , 19387#
fout14 = $8f48: 19391 , 19400#
fout15 = $8f54: 19408#, 19410
fout16 = $8ec9: 19288 , 19294#
```

```
fout17 = $8f6c: 19390 , 19430#
fout19 = $8f69: 19206 , 19425#
fout2  = $8ecf: 19303#, 19323 , 19327 , 19372
fout20 = $8f71: 19420 , 19433#
fout3  = $8e7b: 19238#, 19247
fout37 = $8e65: 19212 , 19218#
fout38 = $8e86: 19242 , 19245#
fout39 = $8eb8: 19276 , 19282#
fout4  = $8e70: 19228#, 19251
fout40 = $8ef5: 19324 , 19329#
fout41 = $8ef3: 19322 , 19327#
fout5  = $8e94: 19243 , 19253#
fout6  = $8ead: 19264 , 19271#
fout7  = $8e6e: 19213 , 19223#
fout8  = $8ecb: 19277 , 19299#
fout9  = $8e8d: 19233 , 19249#
foutb1 = $8f7f: 19450#
foutb2 = $8f83: 19455#
foutb3 = $8f87: 19460#
foutb4 = $8f8b: 19465#
foutb5 = $8f8f: 19470#
foutb6 = $8f93: 19475#
foutb7 = $8f97: 19480#
foutb8 = $8f9b: 19485#
foutbl = $8f7b: 14297 , 19305 , 19309 , 19313 , 19317 , 19368 , 19371 , 19445#
foutc  = $8e44: 17092 , 19177 , 19188#
foutim = $8ecd: 14298 , 19301#
foutpi = $8eac: 19261 , 19270#
foutyp = $8efc: 19330 , 19335#
fpwr   = $8fbe: 19526#, 25526
fpwr1  = $8fe4: 19544 , 19551 , 19555#
fpwrt  = $8fc1:  2899 , 19531#, 25527
fpwrt1 = $8fca: 19534 , 19539#
fr4    = $948f: 20551 , 20552 , 20570 , 20571 , 20628#
fre    = $8000:  2844 , 15496#
fre01  = $87c2: 17590 , 17593#
fre02  = $87ca: 17535 , 17538 , 17603#
fref10 = $8047: 15516 , 15559#
frefac = $8781:  3634 ,  5362 ,  5787 ,  6624 , 17525#, 21203
frefpg = $800c: 15499 , 15509#
frefst = $803a: 15500 , 15551#
frepla = $87c8: 17594 , 17597#
freqhi =   $01:   974#, 11752$, 25307$
freqlo =   $00:   970#, 11749$, 12574$, 25304$
frerts = $87f0: 17577 , 17580 , 17625 , 17628 , 17637#
frespc =   $37:   150#, 14340$, 17371$, 17493$, 17504 , 17505$, 17509$, 20210$,
                20211$, 20222 , 20223 , 20291$, 20295$, 20344$, 20345$, 20384$,
                20385$, 20458 , 20459
frestr = $877e: 17269 , 17520#
fretmp = $8785:  3641 , 17155 , 17454 , 17459 , 17531#, 20904 , 21714 , 21719
fretms = $87e0:  5491 ,  5502 , 13683 , 17534 , 17624#
fretop =   $35:   149#,  4327 ,  4332 ,  4366 ,  4370 ,  4869$,  4870$,  5430 ,  5436
                 5576 ,  5580 ,  8639 ,  8643 , 15526 , 15553 , 15556 , 17576 , 17579
                17587 , 17588$, 17591$, 17593$, 17595$, 20180 , 20184 , 20225$,
                20226$, 20394 , 20398 , 20461$, 20462$
frmev1 = $77f5: 13798 , 13800#
frmevl = $77ef:  3465 ,  3532 ,  5042 ,  5306 ,  5679 ,  6576 ,  8391 , 13648 , 13755
                13797#, 14205 , 16842 , 17515 , 20782 , 20828 , 21583 , 25558
```

```
frmjmp = $60db:  8183 ,  8197 ,  8322 ,  8360 ,  8390#
frmlin = $5d19:  7344 ,  7512#
frmnum = $77d7:  7775 ,  7814 , 10199 , 11236 , 13755#, 17044 , 17653 , 17669 , 17686
frmt6  = $a286: 23601 , 23605#
fsca10 = $a159: 23371#, 23377
fsub   = $882e: 17718#, 25514
fsubt  = $8831:  2890 , 17723#, 18180 , 19676 , 20550 , 25515
ftf    = $96a0: 21013 , 21028#
fundsp = $47d8:  2840#,  3510 ,  3513
garb1  = $92ec: 20264#, 20268 , 20280
garb2  = $9303: 20265 , 20283#
garba2 = $92ea:  4349 ,  8629 , 15551 , 20241 , 20259#, 25562
garbag = $92da: 20201 , 20205 , 20233#
gbasic = $4000:   703#, 13317$, 23067
get    = $5612:  2754 ,  5837#
getadr = $8815: 10153 , 10174 , 15692 , 15816 , 16577 , 17670 , 17691#, 25512
getagn = $5679:  5926#,  5948
getan1 = $9a79: 21744#, 21747 , 21750
getan2 = $9a94: 21757 , 21770#
getan3 = $9a96: 21775#, 21778
getan4 = $9aa9: 21793#, 21805
getan5 = $9aba: 21795 , 21810#
getan6 = $9ac5: 21815 , 21818#
getand = $9a74:  8950 ,  9624 , 11049 , 21734#
getbyt = $87f4:  3487 ,  5249 ,  5630 ,  5853 ,  5880 ,  6831 ,  8396 , 10107 , 10234
                10436 , 10650 , 10760 , 10995 , 11774 , 12067 , 12232 , 12282 , 12313
                12333 , 12370 , 12598 , 16380 , 16501 , 17199 , 17653#, 17676 , 19998
                20033 , 22599 , 22622 , 23673 , 24416 , 24420
getcmp = $4d06:  3660 ,  3672#
getdef = $7dd2: 15175 , 15310#
geterr = $4d96:  3792#,  3800
getfnm = $8528: 16937 , 16971#, 16985
getn   = $5625:  5841 ,  5852#
getnth = $56fb:  6044 ,  6050#
getnum = $8803: 10842 , 15712 , 17669#
getoff = $a605: 24121 , 24142 , 24425#
getpos = $9ce3:  9125 ,  9167 , 22102 , 22264#
getps1 = $9ce8: 22072 , 22270#
getrts = $92d9: 20174 , 20228#
getsad = $8819: 11237 , 17697#
getspa = $9299: 17306 , 20169#, 24798
getspt = $547d:  5358 ,  5455#
getsrt = $5016:  4317 ,  4322#
getst1 = $500a:  4312 ,  4314#
getstk = $4ffe:  4306#,  6789 ,  7738 ,  8206
gettim = $7a1a: 14288 , 14399 , 14407#
gettty = $5635:  5844 ,  5847 ,  5864#
gettyp = $1154:   541#,  9320$,  9379
getval = $a5f2: 24028 , 24059 , 24083 , 24241 , 24314 , 24335 , 24411#
getvm  = $76c5: 13042 , 13333 , 13444 , 13513#, 13531
getwrd = $8812:  6415 ,  6952 ,  8033 ,  8111 , 10775 , 11949 , 17686#, 22571 , 22705
                22803 , 24430 , 24438 , 25400
gfor   = $98f5: 21428#, 21445
ginlin = $56a6:  5957 ,  5961#
givayv = $793c:  1709 ,  1710 ,  3607 ,  6496 , 14182#, 14376 , 16888 , 21908 , 25509
gldtb1 = $9cca: 22146 , 22235#
gloop  = $9318: 20302#, 20317 , 20389
glop1  = $9351: 20353#, 20358
```

```
go1    = $5a4a:  6821 ,  6831#
go64   = $5a54:  6833 ,  6842#
go64rt = $5a5f:  6843 ,  6851#
goasc  = $8682: 17281 , 17287#
gobadv = $7b79: 14716 , 14722 , 14726 , 14733 , 14739 , 14746#
godel1 = $a431: 23962 , 24001#
gofcer = $882b: 17661 , 17692 , 17699 , 17713#
goflot = $4d27:  3692 ,  3695#
goforn = $4c7d:  3549 ,  3560#
gofuc  = $8685: 17202 , 17292#
gogo   = $7c08: 14860 , 14868#
gomain = $4f4c:  4108 ,  4119 ,  4122 ,  4145#
gomldv = $904e: 19649#, 19655
gomovf = $7a81: 14383 , 14387 , 14395 , 14424 , 14438 , 14472 , 14478 , 14499#
gone10 = $4aad:  3117#,  3166
gone20 = $4aed:  3118 ,  3165#
gone30 = $4af0:  3106 ,  3112 ,  3168#
gonprc = $7973: 14165 , 14246#
goover = $8b14: 18378 , 18401 , 18410#, 18421 , 18431 , 18483
gordfn = $a6bd: 24559 , 24602#
gordy  = $401c:  1664#,  1688
gordy2 = $5120:  4551 ,  4575 ,  4597#,  4637
gosub  = $59cf:  2734 ,  6710#
gosusb = $5a1d:  3145 ,  6710 ,  6788#
gotary = $7d2d: 15125 , 15156#
gotnum = $5453:  5413 ,  5420#
goto   = $59db:  2730 ,  5096 ,  6712 ,  6718#,  6826 ,  6941
goto_1 = $59e2:  3146 ,  6723#
gowoto = $5a3d:  3273 ,  6818#
gpbase = $2000:   702#, 10605 , 10625$
gplo10 = $9c0b: 22055 , 22058#
gplo20 = $9c16: 22060 , 22063#
gplot  = $9bfb:  9697 ,  9728 , 21996 , 22050#, 25547
graper = $6bc1: 10664 , 10689 , 10700 , 10746#
graphm =   $d8:   332#,  3783$,  9131 ,  9141 ,  9156 ,  9201 ,  9222 ,  9372 ,  9412
                 9427 ,  9516 ,  9739 ,  9809 ,  9867 , 10025 , 10127 , 10611 ,
                10644$, 10667$, 10672 , 10737 , 10739$, 12777$, 12866$, 15874 ,
                22077 , 22153 , 22298 , 22344 , 22628
grapnt =   $8c:   296#,  9126 ,  9171 ,  9966$,  9974$,  9977$,  9980$,  9983$,
                10082$, 10559$, 10562$, 10564$, 10568$, 13427$, 13436$, 13456 ,
                13458$, 13514$, 13519$, 13535$, 13600$, 13601$, 13618$, 22075 ,
                22090$, 22107 , 22144$, 22147$, 22170 , 22173$, 22188 , 22191$,
                22198 , 22201$, 22224$, 22273$, 22280$, 22283$, 22286$, 22288$
grbg99 = $92e7: 20237 , 20254#
grbpnt =   $50:   168#,  2175 , 20290$, 20294$, 20368 , 20369 , 20471 , 20473 ,
                20477$, 20478$
grbtop =   $61:   200#, 20289$, 20293$, 20333$, 20337$, 20355$, 20363 , 20487 , 20489
                20493$, 20494$
grease = $7da1: 15255 , 15259#
greatr = $1148:   526#, 21967$, 21989 , 21998 , 22018 , 22028
grpcol = $9e2f:  8590 ,  8792 ,  9481 , 22607#
grph10 = $6b69: 10636 , 10650#
grph20 = $6b82: 10668 , 10678#
grph30 = $6b86: 10681#, 10741
grph40 = $6b89: 10673 , 10679 , 10683#, 10733
grph50 = $6bae: 10718 , 10725#
grph60 = $6bb4: 10663 , 10732#
grphik = $6b5a:  2766 , 10635#
```

```
grpos3 = $9d0e: 22300 , 22303#
grprts = $9d1b: 22265 , 22308 , 22315#
gsh010 = $65ac:  9313 ,  9320#
gsh020 = $65b9:  9325 ,  9334#,  9342
gsh030 = $65da:  9353#,  9476
gsh040 = $65f0:  9365#,  9473
gsh050 = $6608:  9373 ,  9379#
gsh060 = $6616:  9382 ,  9392#
gsh070 = $661b:  9381 ,  9396#
gsh080 = $6625:  9398 ,  9408#
gsh090 = $6627:  9387 ,  9393 ,  9397 ,  9410#
gsh100 = $662f:  9413 ,  9417#
gsh110 = $663c:  9421 ,  9424#
gsh120 = $6647:  9428 ,  9433#
gsh130 = $6658:  9444#,  9451
gsh140 = $666f:  9454 ,  9457#
gsh150 = $6683:  9439 ,  9470#
gsh160 = $668b:  9465 ,  9471 ,  9476#
gshape = $658d:  2771 ,  9295#
gtang1 = $9a77:  9543 , 11101 , 21739#, 22710
gtbytc = $87f1:  5741 , 17648#
gtff5  = $a61a: 24427 , 24438#
gtlt8k = $69c4: 10146 , 10167 , 10198#
gtsprn = $6cbb: 10875 , 10995#, 11006
gtv15  = $a602: 24413 , 24420#
hares  = $4023:  1641 ,  1673#
havstr = $6441:  9061 ,  9068#
header = $a267:  2785 , 23584#
help   = $5986:  2778 ,  6649#
help01 = $59a9:  6651 ,  6660 ,  6671#
helper =   $55:   177#,  4639 ,  6662$,  6669$,  6692$
helpsb = $59ac:  4645 ,  6676#
hexd   = $8142:  2870 , 15808#
hexit  = $816b: 15824 , 15827 , 15840#
highds =   $5a:   185#,  3994$,  3995$,  4037 ,  4039$,  4041$,  4048$,  4054$,
                 4057$,  7083$,  7084$,  7100 ,  7103 ,  7530$,  7536$,  7573$,
                 7592 ,  7596 , 14784$, 14785$, 14789 , 14790 , 14796$, 14797$,
                14802 , 14803 , 14828 , 14829$, 14833 , 14834$, 14859 , 14862 ,
                14995 , 14997$, 15000$, 15007$, 15013$, 15016$, 20284$, 20313$,
                20320 , 20324$, 20402 , 23503 , 23505$, 23506$
hightr =   $5c:   190#,  2139 ,  2145 ,  3984$,  3985$,  4015 ,  4020 ,  4029 ,
                 4032$,  4034$,  4056$, 14776$, 14777$, 14878$, 14883$, 14890$,
                14895$, 14966 , 14972 , 14984 , 14987$, 14990$, 15015$
hilink = $01fd:   449#,  1753$
hop_1  = $5c35:  7299#,  7313 ,  7322
hulp   =   $78:   259#,  8467$,  8531$, 11276$, 11281 , 11291$, 20840 , 20859 , 20928
                21015$, 21088$, 21098$, 21105$, 21118$, 21204$
hup    = $9682: 20988 , 21009#, 21060 , 21070 , 21073 , 21077
hyo    = $9927: 21458#, 21462 , 21472 , 21484
hyo1   = $994b: 21479 , 21483#, 21491 , 21496
hyo2   = $9924: 21457#, 21465
icrnch = $0304:   458#,  2199
ident  = $a498: 23985 , 24100#
ierror = $0300:   456#,  2006$,  3719
iescex = $0310:   464#,  3347
iesclk = $030c:   462#,  2224
iescpr = $030e:   463#,  4779
ieval  = $030a:   461#, 14081
```

```
if     = $52c5:  2732 ,  5042#
iftr10 = $5308:  5092 ,  5099#
iftr20 = $531a:  5091 ,  5100 ,  5106 ,  5111#
iftrue = $52fe:  5055 ,  5090#
igone  = $0308:   460#,  3100
ilin10 = $5c56:  7286 ,  7303 ,  7330#,  7376
ilin15 = $5c7e:  7355 ,  7357#
ilin20 = $5c89:  7375#,  7488 ,  7507
illval = $9e4f: 22624 , 22631 , 22643#
imain  = $0302:   457#,  3832
imbd10 = $5c06:  7241 ,  7250#
imbdln = $5bfb:  7186 ,  7190 ,  7239#
in2cor = $9e52:  8593 ,  8798 ,  9072 ,  9307 ,  9486 , 22658#
incfac = $894e: 17985#, 18715
incfrt = $895c: 17980 , 17986 , 17989 , 17992 , 17995#
inchr  = $90e5:  4213 , 19785#
incol2 = $9e4c: 22615 , 22618 , 22629 , 22636#
incolr = $9e32:  9692 ,  9734 , 22612#
incor3 = $9e6d:  8795 ,  9069 , 22682#
incord = $9e70:  9714 , 10101 , 22663 , 22687#
incr2c = $9e60: 22669#, 22675
incrnd = $8c4f: 18715#, 19640
incval = $9f3d: 21797 , 21801 , 22838#
incy   = $96fd: 21090#, 21099
ind1_0 = $4305:  2187#,  3953 ,  4162 ,  4166 ,  4171 ,  7672
ind1_1 = $03b7:   471#,  5411 ,  5556 ,  5561 ,  5565 ,  5574 ,  5795 ,  8014 ,  8727
                 9832 ,  9908 , 11287 , 14506 , 14512 , 14519 , 14523 , 14527 , 14819
                14823 , 14827 , 14832 , 14845 , 14869 , 14875 , 14881 , 15623 , 17284
                17492 , 17604 , 17608 , 17612 , 18326 , 18330 , 18334 , 18338 , 18349
                19103 , 19128 , 20339 , 20379 , 20862 , 24158 , 24363 , 24716
ind2_0 = $03c0:   472#,  7693 , 22944 , 23073
inddef = $42ce:  1717 ,  2121#, 17003 , 17008 , 17031 , 17035
inddpt = $42d8:  2133#, 17128 , 17142 , 17179 , 17211
index1 =   $24:   137#,  2091 ,  2187 ,  2390$,  2394$,  2398 ,  2406$,  2407$,  2420
                 2429 ,  2443 ,  2444$,  2447$,  2454 ,  3908$,  3911$,  3944 ,
                 3946$,  3959$,  4017$,  4031 ,  4038 ,  4153$,  4154$,  4178 ,
                 4182$,  4184 ,  4187$,  4191$,  4192$,  4199 ,  4200 ,  4492$,
                 4504$,  4506$,  4511 ,  4684$,  4685$,  4694 ,  4696$,  4698$,
                 4708 ,  5499$,  5500$,  5511$,  5515$,  5521$,  5524$,  5534$,
                 5538$,  5589$,  5590$,  5596 ,  5597$,  5600$,  7645$,  7648$,
                 7663$,  7665$,  7676$,  7706 ,  7711 ,  7935 ,  7936 ,  8414$,
                 8416 ,  8633$,  8636$,  8714 ,  8718 ,  8722 ,  8724$,  8725$,
                 8754 ,  8759 ,  8775$,  8778$,  8780$,  9804$,  9807$, 13915$,
                13918 , 14005$, 14009$, 14031 , 14502$, 14503$, 14815$, 14816$, 14850
                14851$, 14853$, 14858 , 14902 , 14903$, 14906$, 14969$, 14986 , 14996
                15242 , 15368 , 15413$, 15575 , 15578 , 17162 , 17163$, 17167$, 17383
                17480$, 17481$, 17531$, 17532$, 17550$, 17554$, 17562 , 17564 ,
                17568$, 17569$, 17615$, 17616$, 18282$, 18284$, 18287 , 18291 ,
                18295 , 18299 , 18310 , 18319$, 18320$, 18587$, 18588$, 18591 ,
                18595 , 18599 , 18603 , 18610 , 18634$, 18635$, 18639$, 18643$,
                18647$, 18653$, 18657$, 19123$, 19125$, 20068 , 20069 , 20188$,
                20189$, 20196 , 20216$, 20220$, 20364$, 20426$, 22889$, 22892$,
                22932 , 22934$, 22935$, 22945$, 23061$, 23065$, 23074$, 23079$,
                24375$, 24377$, 24386 , 24387 , 24706$, 24709$
```

index2 = $26: 138#, 2100 , 3061$, 3064$, 3075 , 3078$, 3080$, 3792 ,
3914$, 3925$, 3941$, 3954$, 3960$, 7651$, 7654$, 7684 ,
7686$, 7687$, 7696$, 7708 , 7712 , 7866$, 7867$, 7891 ,
7896 , 7917$, 7922$, 7927 , 9301$, 9302$, 9335 , 9358 ,
10002$, 10006$, 10018 , 15609$, 15612$, 15620 , 15629$, 15630 ,
15744 , 15765 , 18789$, 18790$, 18793 , 18799 , 18806 , 18813 ,
18819 , 18829 , 22897$, 22900$, 22939 , 22941$, 22942$, 23062$,
23068$, 23080$, 23085
indfmo = $42e7: 2151#, 5435 , 5456 , 5463 , 5478 , 6584 , 13686 , 13690 , 13694
14368 , 14372 , 17441 , 20794 , 20799
indfrm = $42d3: 2127#, 20809 , 21198 , 21299 , 21412
indgrb = $42fb: 2175#, 20309 , 20332 , 20336 , 20354 , 20415 , 20425
indice = $66: 211#, 3478 , 3481 , 9251$, 9254$, 13638$, 13641$, 13681 , 13682
14254$, 14255$, 14278 , 14282 , 14381 , 14385 , 14499 , 14500 ,
17192$, 17220 , 17223 , 17297 , 17298 , 17402$, 17404$, 17422 ,
17425 , 17525 , 17526 , 21575 , 21578 , 21586 , 21589
indlo0 = $42ec: 2157#, 3867 , 3907 , 3917 , 4428 , 4433 , 4446 , 4459 , 4463
4546 , 4550 , 4560 , 4564 , 4585 , 4589 , 4647 , 4723 , 4747
7873 , 7878 , 7884 , 7916
indlo1 = $4300: 2181#, 14656 , 14661 , 15118 , 15123 , 15128 , 15134 , 15168 , 15310
15327 , 15338 , 15415 , 15419
indlop = $7cb5: 15044#, 15084
indst0 = $42f1: 2163#, 17331 , 17368
indst1 = $42f6: 2169#, 17438 , 17467 , 17471 , 17475
indtxt = $03c9: 473#, 3190 , 3202 , 3206 , 3210 , 5028 , 5064 , 5076 , 5103
5132 , 5138 , 5142 , 5213 , 5233 , 6243 , 6249 , 6253 , 7113
7120 , 7126 , 7162 , 7166 , 7199 , 7639 , 7959 , 8067 , 8073
8077 , 8088 , 8092 , 8282 , 8288 , 8292 , 14229 , 19099
inhtr0 = $42dd: 2139#, 4047 , 4053
inhtr1 = $42e2: 2145#, 14888 , 14893 , 15006 , 15012
ini = $979f: 20832 , 20891 , 21203#
inim10 = $417f: 1912#, 1915
inimmu = $417a: 1653 , 1673 , 1909#
inis10 = $406c: 1718#, 1721
inis20 = $40a5: 1746#, 1748
inis30 = $40f0: 1815#, 1819
inis40 = $40fc: 1825#, 1827
inisto = $4045: 1675 , 1693#
initst = $0a04: 501#, 1678 , 1680$
iniv10 = $4141: 1870#, 1873
iniv20 = $414c: 1878#, 1881
iniv30 = $416c: 1897#, 1901
iniv40 = $4176: 1903#
inivec = $4251: 1674 , 2004#
inivlp = $4253: 2005#, 2009
inivoi = $4112: 1655 , 1829 , 1844#
inlin = $4f93: 3840 , 4211#, 5961
inlinc = $4f95: 4213#, 4224
inlind = $4f9f: 4215 , 4220#
inloop = $56b8: 5982#, 6192
inlop0 = $56c1: 5987#, 5994
inlop1 = $56e3: 6009 , 6024#
inlop2 = $56e6: 6019 , 6025#
inlpn1 = $7e01: 15342 , 15356#
inlpn2 = $7e00: 15334 , 15354#
inlpnm = $7ddb: 15319#, 15373
inpco1 = $56b2: 5871 , 5977#
inpcom = $5405: 5356#, 6105 , 13643

```
inpcon = $56b0:  5945 ,  5976#
inpflg =   $13:   119#,  5977$,  5999 ,  6065 ,  6126 ,  6200
inpptr =   $45:   157#,  5979$,  5980$,  5990 ,  6176$,  6197 ,  6198 ,  6214
inprt  = $8e26:  3820 , 19159#
inprts = $57c9:  6216 ,  6219 ,  6227#
input  = $5662:  2726 ,  5908#
inputn = $5648:  2725 ,  5880#
insflg =   $f6:   360#,  6894$
inst1  = $99e9: 21597 , 21603#
inst2  = $99f3: 21606 , 21613#
inst3  = $99f8: 21619#, 21622
inst4  = $9a02: 21627#, 21638
inst5  = $9a19: 21648#, 21690
inst6  = $9a2a: 21659 , 21665#, 21696
inst7  = $9a4b: 21687 , 21695#
instfd = $9a4f: 21667 , 21701#
instgo = $4c80:  3472 ,  3565#
instnf = $9a54: 21643 , 21656 , 21660 , 21708#
instr  = $99c1:  3565 , 21575#
insy   = $95fa: 20923 , 20928#, 20934 , 20950 , 20953
int    = $8cfb:  2841 , 18886#, 19546 , 19650 , 20546 , 25523
intadh = $127c:   650#,  3125 , 12251$
intadl = $1279:   649#,  3123 , 12254$
integr =   $09:    85#, 18902$, 19554 , 19652
interr = $7ac0: 14556#, 14603
intflg =   $10:   110#,  5300 ,  6114 ,  6498 , 14362 , 14561$, 14608$, 15030 ,
                15092$
intidx = $84a7: 15052 , 16841#
intrts = $8d21: 18888 , 18915#
intval = $127f:   652#,  1741$,  3108 ,  3137 ,  3142$,  3149 ,  3154$,  6893$, 12256
                12266$, 25137 , 25171
invert = $9d9e: 21853 , 22474#
inz    = $9723: 21108 , 21115#
inz2   = $9725: 21112 , 21116#
iodone = $5658:  5625 ,  5893#,  5940
ioerr3 = $a3b5: 23850 , 23855#
iorele = $565a:  5874 ,  5898#
iqplop = $0306:   459#,  4650
iram0  = $039f:   469#,  2067$,  2140 ,  2158 ,  2164 ,  2188 ,  6215
iram1  = $03ab:   470#,  2078$,  2122 ,  2128 ,  2134 ,  2146 ,  2152 ,  2170 ,  2176
                 2182 ,  3673 ,  3677 ,  5545 ,  6581 ,  6617 ,  9336 ,  9359 , 13614
                14323 , 14339 , 14449 , 14463 , 15584 , 17015 , 21628 , 21634 , 21677
                21684 , 23613 , 23649
irqend = $a9f0: 25229 , 25340#
isary  = $7cab: 14631 , 15025#
isds0  = $79e0: 14332 , 14349#
isds1  = $79c1: 14320#, 14326
isflot = $79fd: 14363 , 14381#
isfn10 = $4c01:  3445#
isfn20 = $4c05:  3445 ,  3448#
isfn30 = $4c16:  3453 ,  3464#
isfun  = $4bf7:  3438#, 14199
isgrap = $a074:  9052 ,  9295 ,  9681 ,  9855 , 10096 , 10461 , 22607 , 23135#
isgrer = $a079: 23136 , 23143#
isletc = $7b3c: 14101 , 14551 , 14568 , 14579 , 14681#, 16333
islrts = $7b45: 14682 , 14688#
isnum  = $79ea: 14261 , 14362#
isp    = $997e: 21464 , 21500 , 21516#
```

```
ispl   = $9892: 21327 , 21363#
ispl1  = $988d: 21330 , 21360#, 21414
issec  = $7ad3: 14566 , 14571#
istpky = $4bb5:  3178 ,  3369#,  4556 ,  8733 , 10800
isvar  = $7978: 14106 , 14253#, 14698
isvds  = $79b4: 14271 , 14307#
ity    = $9704: 21091 , 21095#
izero  = $42cb:  2116#
jchker = $58fe:  6481 ,  6523#
jdel1  = $a43a: 24008#, 24014 , 24020 , 24035
jerro1 = $5761:  6141 ,  6153#
jerro2 = $5819:  6244 ,  6313#
jerror = $9169: 19884 , 19917#
jerxit = $91ab: 19875 , 19948 , 19965#
jfcer1 = $5b35:  7060#,  7104
jfcerr = $5972:  6609 ,  6613 ,  6629#
jmper  =   $56:   178#,  1694$,  3511$,  3514$,  3516 , 17235$, 17249 , 19641$, 19673
joy    = $8203:  2867 , 16052#
joy10  = $821d: 16077#, 16084
joy20  = $823c: 16117 , 16123#
joy30  = $823f: 16055 , 16128#
joytab = $8242: 16110 , 16131#
jprint = $c00c:   710#,  9827 ,  9834 ,  9848
jwind  = $c02d:   711#, 12672 , 12677
kbasin = $ffcf:   750#, 20109
kbotcl = $ff53:   728#, 12750
kbsout = $ffd2:   751#, 15541 , 20115
kc64md = $ff4d:   726#,  6848
kchkin = $ffc6:   747#, 19812 , 23219 , 24831
kckout = $ffc9:   748#, 19795
kclall = $ffe7:   758#, 20133
kclose = $ffc3:   746#, 20127
kclrch = $ffcc:   749#,  1652 , 20121
kclsal = $ff4a:   725#, 23411
kdmacl = $ff50:   727#, 25428
key    = $610a:  2793 ,  8434#
keyd   = $034a:   465#,  4137$,  4142$
keydat = $619d:  8556 ,  8575#
kgetin = $ffe4:   757#, 12855 , 12973 , 13285 , 19821
kicmpv = $ff7a:   739#
kifetv = $ff74:   737#, 10858 , 15697
kistav = $ff77:   738#, 15719
kjmpfr = $ff71:   736#
kjsrfr = $ff6e:   735#,  6444 , 12740
klkupl = $ff59:   730#, 23694
kloads = $ffd5:   752#, 19870 , 23554
kloop  = $43cc:  2255 ,  2312 ,  2366 ,  2387#
kloop2 = $43da:  2397#,  2400
kopen  = $ffc0:   745#, 19772
kpfkey = $ff65:   733#,  8421
kphenx = $ff56:   729#,  1687
kplot  = $fff0:   760#, 20152
kprimm = $ff7d:   740#, 20139
krdtim = $ffde:   755#, 14407
kreads = $ffb7:   742#, 14428 , 20091
ksaves = $ffd8:   753#, 19842
kscror = $ffed:   759#,  9744
ksetnm = $ffbd:   744#, 20103
```

```
kstbnk = $ff68:   734#, 20145
kstlfs = $ffba:   743#, 20097
kstmsg = $ff90:   741#,  3825 ,  6903
kstop  = $ffe1:   756#, 20158
ksttim = $ffdb:   754#,  5406
kswapr = $ff5f:   732#, 10479 , 10499 , 10526 , 10534 , 10545 , 10553 , 10681
kychtb = $61a3:  8503 ,  8580#
kydmsg = $ae63: 15537 , 25471#
lastpt =   $19:   135#,  1727$, 17410$, 17624 , 17627 , 17633$
lcra   = $ff01:   721#,  2039$,  2068$,  2098$,  2107$
lcrb   = $ff02:   722#,  2079$,  2089$
lcrc   = $ff03:   723#,  2044$,  2072$,  2101$,  2110$,  3724$,  4247$,  6366$,
                 8016$,  8208$,  8630$,  8728$,  8776$,  9055$,  9123$,  9195$,
                 9233$,  9298$,  9338$,  9361$,  9784$,  9909$,  9984$, 11272$,
                11288$, 13616$, 14482$, 14491$, 20150$, 20915$, 21307$, 21320$,
                21394$, 21709$, 21715$, 24159$, 24717$, 24920$
lcrd   = $ff04:   724#,  2083$,  2092$,  5329$,  5343$,  5510$,  5533$,  6353$,
                 8774$,  9193$,  9231$, 14914$, 15189$, 15759$, 15823$, 17021$,
                17070$, 17111$, 17369$, 17548$, 20215$, 20274$, 20882$, 24805$,
                24837$, 24911$
ld100  = $8bb4: 18512 , 18554#
ldform = $9536: 20794#, 20797
ldtb1  = $c04c:   713#,  9968 , 13516 , 22275
ldtb2  = $c033:   712#,  9965 , 13513 , 22143 , 22272
ldvn   = $96e1: 21056 , 21072#
ldzr   = $7b52: 14696 , 14704#, 14712
leftd  = $85d6:  2860 , 17125#
len    = $8668:  2855 , 17263#
len1   = $866e: 15564 , 15608 , 17263 , 17269#, 17280
lenchk = $a5dd: 24365 , 24376 , 24382#
lentmp =   $79:   263#, 14872$, 14877
lesser = $1147:   525#, 21964$, 21971 , 22013
let    = $53c6:  2729 ,  3301 ,  5293#,  7727
lfor   = $0135:   443#, 20800$, 21192
linadd = $5d75:  7575 ,  7592#
lincnt = $12fb:   694#, 13332$, 13369
linget = $50a0:  3857 ,  4475#,  5276 ,  6636 ,  6718 ,  7029 ,  7343 ,  7966 ,  7978
                25561
lingrt = $50e1:  4483 ,  4529#
lingt1 = $50bb:  4495 ,  4502#
lingt2 = $50db:  4522 ,  4525#
lininc = $5d80:  7223 ,  7603#
linnum =   $16:   133#,  3124$,  3126$,  3753$,  3757$,  4073 ,  4076 ,  4112 ,  4117
                 4436 ,  4449 ,  4477$,  4478$,  4491 ,  4502 ,  4508 ,  4509$,  4512
                 4513$,  4515$,  4516$,  4518 ,  4520$,  4523$,  4566 ,  4569 ,  6417
                 6420 ,  6638 ,  6641 ,  6656$,  6657$,  6665 ,  6666 ,  6729 ,  6732
                 6954$,  6955$,  7036 ,  7037 ,  7071$,  7074$,  7091$,  7092$,
                 7196$,   208 ,  7532 ,  7538 ,  7548 ,  7555 ,  7558 ,  7987$,
                 7988$,  8112$, 10857 , 15685 , 15688 , 15695 , 15696 , 15702$,
                15705$, 15715 , 15716 , 15810 , 15813 , 15822 , 15826 , 15830$,
                15833$, 16568 , 16571 , 16589 , 16593 , 16597$, 16600$, 17706$,
                17707$, 24433 , 24434 , 24677 , 24680
linprt = $8e32:  4609 , 19171#, 23275 , 24945
list   = $50e2:  2748 ,  4540#
list4  = $50e5:  4545#,  4594
list44 = $50f2:  4547 ,  4556#
lkelse = $52ea:  5061 ,  5066 ,  5073#,  5081
lkups  = $ff5c:   731#, 23376
```

```
llar   = $9951: 21478 , 21487#
lnkprg = $4f4f:  3968 ,  4101 ,  4150#,  6926 ,  7933 , 19930 , 19938 , 22967 , 25553
lnkrt1 = $4f8c:  4202 ,  4204#
lnkrts = $4f92:  4167 ,  4206#
load   = $912c:  2740 , 19857#
loadnp = $9133: 19865#, 23459
locate = $6955:  2774 , 10096#
loderr = $9167: 19913#, 23575
lofbuf =   $ff:   365#
log    = $89ca:  2848 , 18114#, 19560 , 25522
log1   = $89d4: 18116 , 18123#
log2   = $89c5: 18109#, 18153 , 18154
logadr = $a447: 23953 , 24025#
logcn2 = $89a1: 18069#, 18142 , 18143
logco0 = $89b1: 18089#
logco1 = $89ac: 18084#
logco2 = $89a7: 18079#
logco3 = $89a2: 18074#
logeb2 = $9005: 19586#, 19632 , 19633
logerr = $89d1: 18115 , 18120#
lolink = $01fc:   448#,  1754$
loop   = $608a:  2780 ,  8312#
loop1  = $a7b7: 24835#, 24845
loop10 = $60bc:  8315 ,  8360#
loop6  = $a4e6: 24158#, 24164
lopfda = $7cf8: 15105#, 15136
lopfdv = $7d04: 15109 , 15117#
lopfn  = $7b1f: 14651 , 14656#
lopfnd = $7b15: 14648#, 14674
loppta = $7d6e: 15211#, 15245
loprel = $780c: 13842#, 13875
lowtr  =   $61:   205#,  2157 ,  2181 ,  3872 ,  3913 ,  3920 ,  3931 ,  3935 ,  4016
                 4021 ,  4065$,  4068$,  4074$,  4078$,  4083 ,  4085$,  4087$,
                 4096$,  4340 ,  4351 ,  4357$,  4425$,  4426$,  4591$,  4592$,
                 6676 ,  6680 ,  6761 ,  6765 ,  6967 ,  6968 ,  7080 ,  7081 ,
                 7099 ,  7102 ,  7863 ,  7864 ,  7885$,  7886$,  7893 ,  7897 ,
                 7921$, 14643$, 14648$, 14672 , 14770$, 14771$, 14916$, 14920$,
                14926$, 14934 , 14937 , 14967 , 14973 , 15105$, 15106$, 15130 ,
                15135 , 15190$, 15197$, 15209$, 15232$, 15236$, 15286 , 15289$,
                15293 , 15294$, 15472 , 15474 , 22866$, 22891
lpabsz = $11e9:   597#, 16273 , 16277$, 25166$
lpnflg = $1278:   648#,  1740$, 25183$
lpoper = $77fa: 13817#, 13986
lpord  = $11ea:   598#, 25169$
lsg    = $9623: 20943 , 20956#
lsloop = $6141:  8494#,  8532
lster2 = $59b5:  6682 ,  6685#
lster3 = $59cb:  6699 ,  6702#
lstert = $59ce:  6686 ,  6689 ,  6690 ,  6705#
lstest = $617e:  8465 ,  8547#
lstky0 = $6121:  8434 ,  8460#
lstky1 = $6125:  8463#,  8548
lstky2 = $6131:  8473#,  8484
lstky3 = $6139:  8475 ,  8481#
lstky5 = $614a:  8503#,  8507
lstky6 = $615f:  8513 ,  8524#
lstky7 = $6164:  8514 ,  8527#
lstky8 = $6177:  8535 ,  8543#
```

```
lstnd  = $616a:  8531#,  8570
lstpnt =   $4b:   160#,  4260 ,  4271 ,  4603$,  4613 ,  4676$,  4726$,  4750$,
                 5294$,  5295$,  5330$,  5334$,  5341 ,  5342 ,  5356 ,  5509 ,
                 5514 ,  5520 ,  5523 ,  5546$,  5983$,  5984$,  6300$,  6301$,
                 6477$,  6478$,  6543$,  6544$,  6580 ,  7251$,  7254$,  7520 ,
                 7523 ,  7833 ,  7836 ,  9257$,  9260$, 10843$, 10850$, 10860 ,
                10861 , 12942$, 12948$, 12953 , 13066 , 13099$, 13114$, 13178$,
                13309$, 13340 , 13552$, 13553$, 13558 , 13604$, 13607$, 13613 ,
                13634$, 13635$, 19958 , 19984$, 19991 , 20034$, 20043$, 20046 ,
                20057 , 20058
lstspc = $6183:  8504 ,  8555#
luk4it = $59ff:  6742 ,  6745 ,  6756#
lukall = $59fb:  3765 ,  6733 ,  6750#,  8115
main   = $4dc3:  3832#,  3846 ,  4145 , 19932
main00 = $4dd5:  3846#,  3974
main1  = $4de2:  3848 ,  3857#
main10 = $4e0b:  3885 ,  3892#
main2  = $4f03:  4086 ,  4089#
main30 = $4e1a:  3876 ,  3893 ,  3897 ,  3906#
makcr1 = $765b: 13452 , 13454 , 13456#
mapio  = $a845:  1909 ,  6847 ,  8420 ,  9833 ,  9895 , 10266 , 10386 , 10577 , 10616
                10904 , 10971 , 11459 , 11498 , 11540 , 11745 , 12287 , 12300 , 12358
                12543 , 12749 , 12771 , 12805 , 12850 , 12936 , 13280 , 13705 , 13726
                15931 , 16062 , 16155 , 16290 , 16396 , 16445 , 16754 , 19771 , 19794
                19811 , 19820 , 19841 , 19865 , 19956 , 20090 , 20096 , 20102 , 20108
                20114 , 20120 , 20126 , 20132 , 20138 , 20144 , 20157 , 23218 , 23375
                23410 , 23693 , 24971#, 25401 , 25407 , 25413 , 25422
match  =   $64:   210#, 21649$, 21665 , 21682 , 21695$
mid2   = $862d: 17196 , 17201#
midd   = $861c:  2862 , 17191#
midln  =   $77:   253#,  6565$,  6598
midpt  =   $78:   260#,  6552$,  6592 ,  6608 ,  6615
mint   = $9872: 21339 , 21345#
mldexp = $8aee: 18372#, 19687
mldvex = $8b09: 18399#, 19649
mlex10 = $8dd2: 19071 , 19085#
mlexmi = $8df0: 19075 , 19108#
mloop  = $4e5b:  3945 ,  3953#,  3957 ,  3962
mltpl1 = $8a5b: 18224 , 18231 , 18239#
mltpl2 = $8a5e: 18242#, 18275
mltpl3 = $8a7a: 18243 , 18267#
mltply = $8a55: 18212 , 18215 , 18218 , 18221 , 18231#
mltus1 =   $77:   251#,  8406$,  8417 ,  8468$,  8479 ,  8546 ,  9539$,  9542 ,
                 9989$,  9995$, 10000$, 10004 , 10031$, 10038 , 10044 , 10060 ,
                10068 , 10070 , 10076 , 10247$, 10269 , 10384$, 10392 , 10398 ,
                10903 , 10969 , 11000$, 11051 , 11141 , 11273$, 11282 , 12340$,
                12347 , 12355 , 20823$, 20898$, 21076$, 21115$, 21195 , 21228 ,
                21262$, 21265$, 21295$, 21367 , 21388$, 21451 , 22168$, 22172 ,
                22186$, 22190 , 22707$, 22709
mmucr  = $ff00:   720#,  1799 ,  1800 ,  1805 ,  1806 , 18322 , 18353$, 22211$,
                24981$
mmucrl = $d500:  1101#
mmumcr = $d505:  1152#
mmup0h = $d508:  1227#
mmup0l = $d507:  1226#
mmup1h = $d50a:  1232#
mmup1l = $d509:  1231#
mmurcr = $d506:  1193#
```

```
mmutbl = $4189:  1912 ,  1922#
mmuver = $d50b:  1236#
mode   =   $d7:   327#,  1973 ,  3780 ,  6698 , 10314 , 10471 , 10522 , 10541 , 10678
                 10732 , 12604 , 12632 , 15900 , 16004 , 16639
modes  = $6bc4: 10666 , 10751#
monit  = $b000:   709#,  2794
morco1 = $55af:  5726#,  5727
moren1 = $7c9b: 14999 , 15001 , 15009#, 15018
morlin = $50a8:  4483#,  4526
morsts = $4bae:  3193 ,  3361#
mout   = $987b: 21347 , 21350#
mov00  = $93dc: 20474 , 20477#
mov01  = $93eb: 20490 , 20493#
mov1f  = $8bfc: 18626#, 19696 , 20586
mov2f  = $8bf9: 18620#, 19710
movaf  = $8c38:  5385 , 18415 , 18449 , 18687#, 19054 , 19522 , 20539 , 20545 , 25544
movafl = $8c3d: 18693#, 18697
movch0 = $691d: 10019 , 10025#
movch1 = $6938: 10034 , 10055#
movch2 = $6940: 10056 , 10064#
movch3 = $6949: 10064 , 10076#
movch4 = $694c: 10026 , 10046 , 10052 , 10061 , 10071 , 10082#
movchr = $6913: 10012#, 10085
movdo  = $8763: 17170 , 17455 , 17486#
movedn = $5dc6:  7495 ,  7664 ,  7670#
movef  = $8c3b: 18692#, 19643
moveup = $5ddf:  7465 ,  7692#
movfa  = $8c28:  3594 , 17751 , 18666#, 25543
movfa1 = $8c2a: 18671#, 19555
movfal = $8c2e: 18674#, 18678
movfm  = $8bd4:  6326 ,  7805 , 14116 , 16775 , 18461 , 18587#, 19526 , 20599 , 25541
movfnd = $5050:  4394#,  4962 ,  6316 ,  7745 ,  8334
movfr  = $8bc1: 18226 , 18570#
movfrm = $7a85: 14502#, 25540
movini = $5da7:  7463 ,  7493 ,  7644#
movins = $874e:  5487 , 17450 , 17466#
movlp  = $8767: 17491#, 17496
movlpt =   $0d:    98#,  7657$,  7670 ,  7692
movmf  = $8c00:  5344 ,  6354 , 16813 , 17023 , 18632#, 19541 , 20595 , 25542
movpnt = $93d2: 20310 , 20348 , 20388 , 20412 , 20469#
movrts = $8c46: 18701#, 18707 , 18710 , 18716
movrvs = $6934: 10039 , 10051#
movs05 = $6cd4: 11016 , 11025#
movs10 = $6cfe: 11062#, 11066
movs15 = $6d08: 11071#, 11077
movs20 = $6d16: 11082#, 11085
movs25 = $6d24: 11041 , 11098#
movs30 = $6d34: 11112#, 11122
movs35 = $6d49: 11040 , 11135#
movs40 = $6d54: 11130 , 11141#
movs45 = $6d67: 11158 , 11163#
movs50 = $6d6a: 11155 , 11164#
movs55 = $6d82: 11176 , 11179#
movs60 = $6d88: 11170 , 11183#
movs65 = $6d91: 11180 , 11188#
movs70 = $6d99: 11189 , 11193#
movspr = $6cc6:  2803 , 11006#
movspt =   $6d:   237#,  7464$,  7494$,  7658$,  7674 ,  7695
```

```
movtop = $93e1: 20342 , 20382 , 20409 , 20485#
movtos = $5047:  4234 ,  4383#,  7740 , 22997
msir10 = $a8f6: 25189#, 25220
msir15 = $a922: 25190 , 25197 , 25203 , 25217#
msir20 = $a92a: 25223#, 25227
msir25 = $a92f: 25226#, 25323
msir30 = $a935: 25224 , 25232#
msir35 = $a961: 25246 , 25255#
msir40 = $a96f: 25256 , 25264#, 25294
msir45 = $a97e: 25244 , 25273#
msir50 = $a994: 25273 , 25279 , 25287#
msir55 = $a9a2: 25262 , 25288 , 25296#
msir60 = $a9ae: 25253 , 25270 , 25277 , 25283 , 25285 , 25302#
msir65 = $a9c7: 25312 , 25315#
msir70 = $a9d7: 25317 , 25323#
mul10  = $8b17:  5378 ,  5400 , 18415#, 19019 , 19040 , 19245
mul10r = $8b2d: 18417 , 18433#
mulco1 =   $84:   290#,  1769$,  9880 , 10292$, 10397 , 15977
mulco2 =   $85:   291#,  1774$,  9883 , 10298$, 10590 , 15983 , 22223
muldiv = $8aec: 18203 , 18367#, 18481
mulshf = $8962: 18006#, 18236
multrt = $8a88: 18200 , 18277#
musirq = $a8f4: 25103 , 25163 , 25180 , 25188#
mutabl = $6fec: 11349 , 11807#
mvdflg =   $76:   246#,  1730$, 22851 , 22882$, 22974 , 23045 , 23057$, 23110 , 23135
mvdnlp = $5dc0:  7663#,  7679
mvdon1 = $8776: 17507#
mvdone = $8771: 14345 , 17377 , 17487 , 17503#
mvstrt = $877a: 17507 , 17510#
mvup10 = $5ddd:  7685 ,  7687#
mvuplp = $5dd7:  7684#,  7699
mx10_1 = $8de8: 19097 , 19103#
mx10_2 = $8deb: 19100 , 19105#
mxmem0 = $1212:   614#,  1802$,  1803$,  3892 ,  3896 ,  3997 ,  4001 ,  7402 ,  7405
                15510 , 15513 , 22868 , 22872
mxmem1 =   $39:   151#,  1808$,  1809$,  4866 ,  4867 ,  5567 ,  5571 , 20286 , 20287
n0999  = $8e17: 19144#, 19238 , 19239
n200   = $69dd: 10169 , 10170 , 10223#
n320   = $69d8: 10148 , 10149 , 10218#
n32768 = $849a: 16828#, 16861 , 16862
n9999  = $8e1c: 19149#, 19228 , 19229
nam01  = $a42e: 23988 , 23991 , 23996#
name1  = $a4dc: 23996 , 24153#
name2  = $a558: 24230 , 24233 , 24265#
nbr    = $9808: 21279#, 21287
ncrnch = $430d:  2024 ,  2204#
ndx    =   $d0:   325#,  4144$
negfac = $8926: 17841 , 17952#
negfch = $892c: 17959#, 18857
neghlf = $89c0: 18104#, 18146 , 18147
negop  = $8ffa:  2908 , 19029 , 19574#, 19677 , 20568 , 20577 , 20672 , 20699 , 25525
negprc = $784b: 13931#, 14248
negrts = $9004: 19569 , 19575 , 19581#
negstp = $a9da: 25258 , 25290 , 25326#
negxqs = $8d99: 19024 , 19029#
nerror = $4d3f:  2022 ,  3724#
nescex = $4ba9:  2030 ,  3352#
nescfn = $4c78:  2011 ,  2014 ,  3554#
```

```
nesclk = $4321:  2028 ,  2229#
nescpr = $51cd:  2029 ,  4800#
neval  = $78da:  2027 , 14086#
new    = $51d6:  2755 ,  4811#, 25552
newbyt = $1157:   548#,  9363$,  9368$,  9375$
newchr = $5557:  5651#,  5690
newnam = $a5b9: 24154 , 24266 , 24356#
newnum = $5d89:  7140 ,  7603 ,  7616#
newstt = $4af6:  3147 ,  3178#,  3766 ,  6713 ,  6928 ,  6943 , 25556
next   = $57f4:  2723 ,  6276#
next10 = $5809:  6276 ,  6298#
next20 = $580e:  6281 ,  6301#
next30 = $581c:  6305 ,  6316#
next35 = $5829:  6323 ,  6326#
next40 = $5855:  6363 ,  6366#
next45 = $5877:  6291 ,  6392#
next50 = $57fa:  6286#,  6372
next6  = $a503: 24185#, 24211
next6a = $a50a: 24186 , 24195#
next6b = $a510: 24196 , 24202#
nf     = $012d:   435#, 21002 , 21058 , 21224
ngone  = $4aa2:  2026 ,  3105#
nibble = $1239:   635#, 12033$, 12045 , 12122$, 12129 , 12142$, 12149
nlnfac = $5d68:  7109 ,  7245 ,  7512 ,  7581#
nloop  = $4ed1:  4047#,  4051
nmain  = $4dc6:  2023 ,  3837#
nmary1 = $7d15: 15121 , 15127#
nmil   = $8e21: 19154#, 19218 , 19219
no4    = $969d: 21021 , 21027#
nocr   = $95e4: 20907 , 20912#
nodel  = $4e6a:  3862 ,  3967#
nodel1 = $4e89:  3988 ,  3990#
nodel2 = $4ea2:  3998 ,  4002 ,  4011#
nodel3 = $4ec5:  4033 ,  4037#
nodel5 = $4ed6:  4040 ,  4042 ,  4050#,  4059
nodel6 = $4ede:  4027 ,  4056#
nodelc = $4e8f:  3992 ,  3994#
nodele = $4e79:  3972 ,  3979#
noml6  = $5432:  5387 ,  5395#
nomn   = $962d: 20958 , 20961#
nono   = $990d: 21437 , 21442#
nonono = $84c1: 16849 , 16864#
norm1  = $88fc: 17911#, 17917
norm2  = $8908: 17851 , 17917#
norm3  = $88ba: 17850#, 17871
normal = $88b6: 16809 , 17839 , 17846#, 18582
nortok = $51c7:  4769 ,  4777#
nos1   = $9705: 21093 , 21096#
nos3   = $9727: 21023 , 21117#
nos4   = $9708: 21027 , 21097#
nosec  = $7ade: 14569 , 14582#
nosflt = $84c9: 14486 , 15559 , 15675 , 16246 , 16287 , 16359 , 16558 , 16594 ,
                16874#
not500 = $5ef5:  7941 ,  7943#
notabr = $55d4:  5677 ,  5750 ,  5767#
notdel = $5ee5:  7234 ,  7900 ,  7933#
notdim = $7d7e: 15219 , 15231#
noteos = $5ead:  7876 ,  7884#
```

```
notes  = $6fe7: 11321 , 11798#
noteve = $7ba6: 14781 , 14784#
notevl = $7b7c: 14699 , 14749#
notfdd = $7d46: 15112 , 15180#
notfl1 = $7e23: 15382 , 15386#
notfl2 = $7e29: 15387 , 15392#
notflt = $7d5c: 15191 , 15195#
notfns = $7b46: 14654 , 14693#
notit  = $7b32: 14658 , 14671#
notop  = $7930:  2911 , 14170#
notqti = $5671:  5888 ,  5909 ,  5921#
notqut = $5c22:  7265 ,  7278#
notstr = $7ae8: 14583 , 14594#
nowge1 = $572e:  6100 ,  6103#
nowget = $5723:  6086 ,  6094#
nowlin = $57e6:  6236 ,  6258#
nqplop = $5151:  2025 ,  4655#
nrange = $5e84:  7851#,  7859
nstt10 = $4b03:  3181 ,  3189#
nstt20 = $4b0d:  3191 ,  3198#
nstt30 = $4b2e:  3220 ,  3223#
nstt40 = $4b31:  3199 ,  3203 ,  3226#
nsxx6  = $4bf3:  3422 ,  3430#
ntime  = $1229:   626#,  1847$,  1848$, 11566$, 11569$, 11576$, 11577$, 11707 , 11710
                 11722 , 11726
numcnt = $116e:   574#,  9785$,  9829 ,  9911$, 11012 , 11035 , 11124$, 11149$,
                 11169$, 11203$, 11231$, 13337$, 13343
numerr = $5a1a:  6721 ,  6783#
numflg =   $0a:    88#,  4476$,  4487$,  6720 ,  7031
numins = $573a:  6061 ,  6111#
nxsc10 = $aa17: 25370 , 25376#
nxsc20 = $aa1b: 25373 , 25378#
nxsc30 = $aa1e: 25362 , 25380#
nxscor = $a9f4: 25056 , 25064 , 25346#
nxtchr = $5c10:  7259#,  7275 ,  7280 ,  7304 ,  7310 ,  7316
nxtcmp = $4cfb:  3658#,  3682
nxtlin = $5bfe:  7240#,  7271 ,  7279 ,  7300
nxtptr = $7b31: 14663 , 14670#
nxtqut = $5c17:  7270#,  7274
nxtreg = $58eb:  6472 ,  6504#
nxxx   = $a519: 24178 , 24210#
oblk   = $958d: 20855#, 20860
octave = $122b:   627#,  1851$, 11487$, 11598
offtab = $7039: 11504 , 11747 , 11916#, 12542 , 25212 , 25302 , 25320
oindi1 = $42b0:  2089#
oindi2 = $42b9:  2098#
ointxt = $42c2:  2107#
oiram0 = $4298:  2067#
oiram1 = $42a4:  2078#
okgoto = $52d4:  5046 ,  5054#
oknorm = $4c3b:  3456 ,  3459 ,  3499#
oldcl1 = $a820: 24905 , 24919#
oldclr = $a80d: 20082 , 23395 , 24519 , 24864 , 24901#
oldlin = $1200:   602#,  3414$,  3415$,  6878 ,  6879 ,  7331$,  7334$,  7346 ,  7349
oldov  =   $58:   179#, 17755$, 17787$, 17817 , 17887
oldstk =   $82:   288#,  3187$,  3762
oldtxt = $1202:   603#,  3234$,  3235$,  3740 ,  4920$,  6162 ,  6163 ,  6865 ,  6873
                 22989 , 22990$
```

```
oldvol = $1275:   645#,  1864$, 11455$, 11537 , 12350 , 12356$
olgarb = $54d0:  5504 ,  5520#
omerr  = $4d3a:  3715#,  3901 ,  4006 ,  4378 ,  7432 ,  8429 ,  8765 , 15351 , 20254
                 22877
omerr1 = $5cc4:  7406 ,  7432#
omerr2 = $4e9f:  4000 ,  4006#
omerr3 = $6107:  8422 ,  8429#
omerr4 = $7dfd: 15248 , 15257 , 15351#, 15440 , 15459
omerr5 = $4e17:  3894 ,  3901#
omerr6 = $5044:  4316 ,  4320 ,  4368 ,  4371 ,  4378#
omerr7 = $9f6a: 22865 , 22870 , 22877#
on     = $a582: 24007 , 24253 , 24304#
on1    = $a437: 23968 , 24007#, 24190
on2    = $a54d: 24224 , 24253#, 24287
onglop = $53b2:  5256 ,  5259 ,  5266#,  5279
onglp1 = $53ba:  5267 ,  5275#
ongoto = $53a3:  2738 ,  5249#
open   = $918d:  2752 , 19944#
opmask =   $4f:   165#,  3632$, 13835$, 13858 , 13868 , 13873$, 13878 , 13957$, 13985
opptr  =   $4d:   163#, 13934 , 14045$
optab  = $4828:  2886#, 13923 , 13968 , 13975 , 13978 , 13997 , 25545
optbyt = $9e1e:  6428 ,  6432 ,  6436 ,  6440 ,  9583 , 10349 , 10697 , 10879 , 10893
                10910 , 10922 , 10934 , 10946 , 11982 , 12017 , 12112 , 12135 , 12162
                12293 , 12463 , 22592#, 23682
optw10 = $9e17: 22565 , 22569 , 22578#
optw99 = $9e1a: 22581#, 22593 , 22597
optwrd = $9e06:  7045 ,  7065 ,  8802 ,  9490 ,  9496 ,  9526 ,  9532 ,  9538 , 12193
                12247 , 12428 , 12437 , 12490 , 22564#
optzer = $9e1c:  8596 ,  8808 ,  9311 ,  9795 , 10683 , 12409 , 12658 , 22587#
orop   = $4c86:  2905 ,  3575#
osc1   =   $00:  1019#,  1857$, 11758$, 11761$, 11916 , 12549$, 25215$, 25321$
osc2   =   $07:  1020#,  1858$, 11916
osc3   =   $0e:  1021#,  1859$, 11916
oscctl =   $04:   991#,  1857$,  1858$,  1859$, 11758$, 11761$, 12549$, 25215$,
                25321$
out    = $98ab: 21310 , 21314 , 21323 , 21364 , 21379#
outc   = $9599: 20856 , 20864#
outch  = $90df:  5830 , 19778#, 23658
outdo  = $560c:  3255 ,  3260 ,  3796 ,  4619 ,  4711 ,  5707 ,  5713 ,  5796 ,
                5830# ,  6702 , 21422 , 21444
outqst = $560a:  3789 ,  5825#,  5959 ,  6048
outrt  = $98b3: 21380 , 21385#
outs   = $98a8: 21378#, 21397
outspc = $5600:  5689 ,  5776 ,  5807#
overr  = $895d: 17938 , 18000#, 18410 , 19080 , 21150
p1l010 = $512e:  4613#,  4709
p1l015 = $5132:  4619#,  4655 ,  4658 ,  4664 ,  4724 ,  4748 ,  4806
p1l019 = $513f:  4624 ,  4636#
p1l020 = $5149:  4643 ,  4647#
p1l026 = $516a:  4684#,  4739 ,  4763
p1l030 = $5170:  4688#,  4702
p1l040 = $5173:  4694#,  4701
p1l050 = $517c:  4697 ,  4700#
p1l060 = $5181:  4707#,  4712
p1l070 = $5182:  4689 ,  4708#,  4803
p1line = $5123:  4580 ,  4602#,  6667
p1lrts = $518b:  4648 ,  4715#
p2cd10 = $5c9a:  7393#,  7410
```

```
p2cd20 = $5ca0:  7394 ,  7400#
p2cd30 = $5cb4:  7385 ,  7417#,  7425
p2cd40 = $5cbf:  7421 ,  7424#
p2cd50 = $5cc3:  7418 ,  7427#
p2code = $5c8f:  7370 ,  7382#,  7388
p3cd10 = $5cc8:  7439#,  7482
p3cd15 = $5ce1:  7446 ,  7449 ,  7463#
p3cd20 = $5cf1:  7458 ,  7470 ,  7476#
p3cd30 = $5cf9:  7440 ,  7487#
p3cd40 = $5cfe:  7493#,  7505
p3cd50 = $5d0e:  7500 ,  7502#
p3code = $5cc7:  7366 ,  7437#
pain07 = $6265:  8723 ,  8725#
paint  = $61a8:  2767 ,  8590#
paint2 = $61bd:  8598 ,  8605#
paits2 = $62af:  8779 ,  8782#
paoc   = $91f6: 19944 , 19955 , 20026#
paoc15 = $9239: 19978 , 20067#
paoc20 = $91e3: 19977 , 19980 , 19987 , 20003#, 20041 , 20053 , 20061
paoc30 = $91eb: 19997 , 20013#, 20062
paoc32 = $91ee: 20018#, 20032
paoc5  = $9221: 20048 , 20051#
paocx  = $91ea: 20004 , 20008#, 20019
papp   = $a63e: 23351 , 24481#
parchk = $7950:  3499 , 14195 , 14204#, 16993
parcnt = $123e:   638#, 12095$, 12126 , 12146 , 12152
pars22 = $a526: 24204 , 24220#
parse1 = $a3f8: 23926 , 23952#, 24214
parse2 = $a523: 24219#, 24284
parsts =   $80:   271#, 12708 , 20926$, 20949 , 20964 , 20967$, 21019 , 21028 , 21085
                21121$, 21162$, 21213$, 21223 , 21260$, 21270 , 21279$, 23157 , 23169
                23333 , 23399 , 23743 , 23762 , 23769 , 23772 , 23902$, 23938 , 24171
                24172$, 24277 , 24278$, 24370 , 24372$, 24443 , 24631 , 24743 , 24748
                24765 , 24774 , 24784
parstx =   $81:   282#, 12720 , 23472 , 23903$, 23932 , 23939 , 24131 , 24132$, 24342
                24343$, 24451
pas1_1 = $5b74:  7119#,  7131
pas1_2 = $5b8a:  7128 ,  7137#,  7150
pas2_1 = $5bc4:  7192#,  7224 ,  7231
pas2_2 = $5bdf:  7206 ,  7215#
pas2_3 = $5bf0:  7205 ,  7210 ,  7229#
pbadvl = $6efd: 11381 , 11413 , 11472 , 11483 , 11553#
pbak   = $a64b: 23824 , 24493#
pchar  = $9911: 21430 , 21444#
pchi   =   $03:    77#,  6421$,  6486 , 12738$
pclo   =   $04:    78#,  6418$, 12735$
pcoll  = $a649: 23738 , 24489#
pconc  = $a635: 23785 , 24476#
pcra   = $d501:  1137#,  1913$
pcrb   = $d502:  1139#
pcrc   = $d503:  1141#
pcrd   = $d504:  1143#
pdclr  = $a627: 24464#
pdcop  = $a64f: 23774 , 24497#
pdir   = $a629: 23166 , 24468#
pdop   = $a62d: 23330 , 23421 , 23451 , 23492 , 23543 , 24472#
peek   = $80c5:  2854 , 15685#
pen    = $82ae:  2875 , 16256#
```

```
pen10  = $82d0: 16280 , 16286#
pen20  = $82d6: 16269 , 16290#
pen30  = $82e4: 16298 , 16302#
pen40  = $82ed: 16292 , 16314#
pen50  = $82f7: 16321 , 16325#
phed   = $a643: 23598 , 24485#
pi2    = $9485: 20532 , 20533 , 20618#, 20692 , 20693
pitch  = $122d:   629#, 11625$, 11636$, 11687$, 11748 , 11751
pival  = $78fe: 14114 , 14115 , 14123#
pkydef = $100a:   508#,  8494
pkylen = $1000:   507#,  8464
pl1c10 = $6e07: 11302 , 11309#
pl1c15 = $6e12: 11310 , 11313 , 11320#
pl1c20 = $6e14: 11321#, 11328
pl1c25 = $6e1c: 11322 , 11327#
pl1c30 = $6e26: 11333 , 11340#
pl1c35 = $6e2d: 11341 , 11348#
pl1c40 = $6e2f: 11349#, 11356
pl1c45 = $6e37: 11350 , 11355#
pl1c50 = $6e41: 11361 , 11368#
pl1c55 = $6e48: 11369 , 11376#
pl1c60 = $6e52: 11379 , 11386#
pl1chr = $6e02: 11289 , 11301#
play   = $6de1:  2801 , 11271#
playlp = $6dee: 11281#, 11292
playm  = $6f5d: 11644 , 11655#
playml = $6f5f: 11660#, 11661 , 11666
playrt = $6e01: 11283 , 11295#
plcmd  = $6f52: 11352 , 11643#
pldot  = $6f03: 11343 , 11559#
plflat = $6f6c: 11371 , 11679#
plnlp1 = $6f81: 11704#, 11705
plnot1 = $6f37: 11604 , 11611#
plnot2 = $6f3e: 11612 , 11622#
plnot3 = $6f48: 11632#, 11638
plnot4 = $6f72: 11633 , 11687#
plnot5 = $6fae: 11718 , 11736#
plnote = $6f1e: 11315 , 11587#
plot01 = $9c19:  8673 ,  9418 , 22052 , 22058 , 22068#
plot10 = $9c3d: 22089#, 22097
plot20 = $9c40: 22090#, 22096
plot30 = $9c43: 22078 , 22095#
plotgo = $9c42: 22069 , 22092#
plrest = $6f78: 11335 , 11695#
plshrp = $6f69: 11363 , 11673#
plsv   = $91ae: 19828 , 19860 , 19968#
plsv7  = $91dd: 19981 , 19988 , 19997#, 20042 , 20054
plyflg = $0126:   421#, 11386$, 11397$, 11400$, 11403$, 11547$, 11649$
pndd   = $98c8: 21333 , 21400#
pndx   = $98d0: 21404#, 21410
pndy   = $98d5: 21404 , 21409#
pnterr = $61cf:  8615 ,  8621 ,  8624#
pntl   = $9711: 21086 , 21103#
pntok1 = $61c8:  8612 ,  8620#
pntok2 = $61d0:  8622 ,  8629#
point1 = $8318: 16352 , 16359#
pointr = $82fa:  2881 , 14695 , 16330#
poit   = $972e: 21101 , 21121#
```

```
poke   = $80e5:  2744 , 15712#
polcrd = $9e84: 22695 , 22704#
poly   = $909c: 19681 , 19707#
poly1  = $90a0: 19700 , 19710#
poly2  = $90b3: 19723#, 19743
poly3  = $90af: 19717 , 19720#
poly4  = $90c0: 19731 , 19734#
polypt =   $72:   243#, 19693$, 19694$, 19707$, 19708$, 19711 , 19714 , 19718$,
                  19720$, 19722 , 19727 , 19728 , 19734$, 19735$
polyx  = $9086: 18144 , 19693#, 20581 , 20686
ponter = $831b: 16334 , 16364#
popdgo = $609b:  8237 ,  8330#,  8361 ,  8366
poperr = $60b7:  8332 ,  8354#
popngo = $60c1:  8312 ,  8325 ,  8366#
pos    = $84d0:  2845 , 16881#
posint = $84ad: 16847#, 17658
positn =   $63:   207#, 21614$, 21655 , 21671 , 21689$, 21701$, 21702
posp   = $012e:   436#, 20990 , 21046 , 21409 , 21525 , 21528$
post   = $9879: 21343 , 21349#
postab = $11d6:   593#, 11163 , 11165$, 11173 , 11174$, 11183$, 12900$, 12903$, 16553
                25012 , 25369 , 25378$
pot    = $824d:  2873 , 16146#
pot10  = $8264: 16162 , 16168#
pot20  = $8269: 16169 , 16175#
pot30  = $8277: 16191#, 16192
pot40  = $827a: 16197#, 16202
pot50  = $828d: 16209 , 16213#
pot60  = $8293: 16216 , 16225#
pot70  = $829f: 16231 , 16238#
poterr = $82ab: 16153 , 16251#, 16266
potx   =   $19:  1076#, 16197 , 16201
poty   =   $1a:  1080#
prcha  = $9587: 20835 , 20841 , 20850#
pream  = $864d: 17125 , 17176 , 17201 , 17229#
preamb = $a82a:  8473 , 24934#
prec   = $a663: 23706 , 24509#
pren   = $a657: 23797 , 24501#
prescc = $518c:  4670 ,  4720#
prescf = $51a6:  4673 ,  4744#
prforn = $51c0:  4729 ,  4732 ,  4753 ,  4756 ,  4768#
print  = $555a:  2746 ,  5641 ,  5656#
printc = $5563:  5659 ,  5664#,  5768
printn = $553a:  2745 ,  5624#
priopt = $7839: 13893 , 13902#
prmrpt = $a61d: 23936 , 24026 , 24048 , 24081 , 24239 , 24356 , 24368 , 24443#
prtdec = $a830:  8564 , 24939#
prtrts = $55a8:  5664 ,  5701 ,  5716#,  5793
prxrpt = $a622: 24119 , 24140 , 24333 , 24451#
pscr   = $a65f: 23634 , 24505#
ptarg1 =   $59:   181#, 21620$, 21627
ptarg2 =   $5b:   188#, 21633
ptbotm = $6203:  8663 ,  8666#
ptdone = $6279:  8720 ,  8737#
ptdown = $61f1:  8657#,  8664
ptdwn2 = $61f9:  8658 ,  8660#
pterr  = $6297:  8757 ,  8765#
ptfil2 = $6216:  8676 ,  8678#
ptfil3 = $622e:  8688 ,  8691#
```

```
ptfil4 = $623d:  8696 ,  8698#
ptfil6 = $6248:  8701 ,  8707#
ptfill = $620b:  8667 ,  8673#,  8709
ptloop = $61eb:  8650#,  8734
ptmax  = $624f:  8708 ,  8711#
ptmax2 = $625f:  8716 ,  8722#,  8731
ptr2   =   $9f:   316#
ptrget = $7aaf:  5293 ,  5982 ,  6298 ,  6476 ,  6542 ,  9053 , 13633 , 14253 ,
                14535#, 16336 , 16944
ptrgt1 = $7ab4:  6403 , 14541#
ptrgt2 = $7ab6: 14546#, 16976
ptrgt3 = $7ac3: 14552 , 14559#
ptsave = $629e:  8756 ,  8761 ,  8773#,  8784
pucoma = $1205:   605#, 21309
puctrl = $5f34:  2765 ,  8000#
pudefs = $5250:  4890 ,  4929#
pudot  = $1206:   606#, 21322
pufill = $1204:   604#,  4891$,  8018$, 20917
pulmor = $8603: 17165 , 17169#
pulop  = $5f40:  8005 ,  8014#,  8020
pulsec =   $03:   984#, 11516$
pulsef =   $02:   980#, 11513$
pulshi = $1267:   643#,  1879$, 11515 , 12212$
pulslo = $125d:   642#, 11512 , 12215$
pulstk = $78ba: 13939 , 13969 , 14047#
pumony = $1207:   607#, 21396
punt   = $9846: 21302 , 21317#
pushf  = $7884: 13981 , 13996#
putnew = $86e3: 14349 , 17120 , 17171 , 17363 , 17382#, 17460 , 20890
putnw1 = $86ee: 17384 , 17392#
pwhmus = $702f:  1878 , 11902#
qchnum = $78b1: 13924 , 14040#
qcomp  = $4d01:  3624 ,  3667#
qdat1  = $56db:  6016#,  6018
qdata  = $56ef:  6003 ,  6034#
qdata1 = $56f4:  6034 ,  6043#
qdect1 = $4e53:  3939 ,  3943#
qdot   = $7903: 14110 , 14126#
qdsav  = $7a36: 14421 , 14434#
qerlin = $7a60: 14435 , 14471#
qinlin = $569c:  5926 ,  5956#,  6050
qint   = $8cc7:  5401 , 16869 , 17701 , 18845#, 19255 , 25437
qint1  = $8ce9: 18863 , 18871#
qintgo = $84c6: 16859 , 16864 , 16869#
qintgr = $53e3:  5319#,  6115 ,  6499
qintrt = $8ce8: 18831 , 18868#
qishft = $8cdc: 18851 , 18860#
qnum   = $0390:   468#,  5412
qnumer = $7a78: 14475 , 14491#
qop    = $78ac: 13883 , 13888 , 14034#
qopgo  = $78af: 13938 , 14038#
qoprts = $78d4: 14038 , 14075#
qplus  = $8d38: 18940 , 18946#
qprec  = $7841: 13921#, 13965
qprec1 = $786a: 13935 , 13968#
qsetnr = $8455: 16750 , 16773#
qshft  = $8b86: 18508 , 18516#, 18555
qst001 = $7b57: 14711#, 14753
```

```
qst002 = $7b6b: 14731#, 14756
qst003 = $7b71: 14738#, 14759
qst004 = $7b61: 14721#, 14762
qstatv = $7a28: 14392 , 14420#
qtyer2 = $a495: 24030 , 24064 , 24067 , 24085 , 24095#
qtyer3 = $a215: 23484 , 23489 , 23490 , 23516#
qtyerr = $a57f: 24243 , 24299#, 24337
qvaria = $5468:  5432 ,  5439#
r8502  =   $01:   779#, 10580 , 10588$, 10596$, 22214 , 22221$, 22227$
raddc  = $8495: 16781 , 16782 , 16823#
rambot = $1c00:   698#,  1729$, 22905 , 23064
random =   $1b:  1084#
range  = $5efb:  4540 ,  7861 ,  7950#
rbits  = $9d1c: 11648 , 13076 , 13077 , 13091 , 13097 , 22306 , 22313 , 22320#
rclr   = $819b:  2865 , 15921#
rclr10 = $81b9: 15933 , 15960#
rclr20 = $81c1: 15936 , 15971#
rclr30 = $81c6: 15939 , 15977#
rclr40 = $81cb: 15942 , 15983#
rclr50 = $81d0: 15945 , 15989#
rclr60 = $81d6: 15948 , 15999#
rclr70 = $81de: 15951 , 16011#
rclr80 = $81e6: 16006 , 16019#
rclr90 = $81ec: 15966 , 15972 , 15978 , 15984 , 15994 , 16005 , 16023#
rdcn   = $a703: 24547 , 24677#
rdfn   = $a723: 24602 , 24705#
rdmov  = $a734: 24716#, 24724
rdot   = $9b0c:  2868 , 21878#
rdot10 = $9b18: 21881 , 21890#
rdot20 = $9b20: 21892 , 21897#
rdot30 = $9b23: 21880 , 21902#
rdrt0  = $a745: 24688 , 24712 , 24727#
rdrt1  = $a746: 24700 , 24728#
rdy    = $9730: 21083 , 21123#
read   = $56a9:  2728 ,  5966#
readgo = $9c6f: 22103 , 22138#
readl1 = $9c65: 22115 , 22125#
readl3 = $9c6d: 22127 , 22137#
readlp = $9c56: 22109#, 22111
readpt = $9c49:  8620 ,  8662 ,  8707 ,  8744 ,  9365 , 21890 , 22102#
ready  = $4d37:  1665 ,  3226 ,  3430 ,  3709#,  6644 ,  7945 ,  8082
readyx = $4db7:  3727 ,  3818 ,  3822#
realsp = $5604:  5813#,  5960
reas10 = $5021:  4330 ,  4338#
reas20 = $5025:  4344#,  4347
reas30 = $5030:  4356#,  4359
reason = $5017:  4327#, 14958 , 15181 , 15259
reasrt = $5043:  4328 ,  4333 ,  4367 ,  4373#
reay   = $95c7: 20866 , 20893#
reclen = $a45d: 23956 , 23959 , 24045#
reco0  = $a46d: 24051 , 24059#
recon  = $a47b: 24056 , 24074#
record = $a2d7:  2815 , 23670#
reddy  = $4d2a:  3700#,  3822 , 19929
regrts = $58fd:  6469 ,  6508 ,  6513 ,  6518#
rem    = $529d:  2333 ,  2736 ,  5004#,  5077 ,  5172 ,  5239
remer  = $52b5:  5028#,  5036
remn   = $52a5:  5004 ,  5016#,  6723
```

```
remrts = $529c:  4997 ,  4999#,  5029 ,  5032
rename = $a36e:  2789 , 23792#
renum  = $5af8:  2792 ,  7008#,  7284
repend = $596f:  6605 ,  6624#
repl10 = $591c:  6557 ,  6564#
repl20 = $5930:  6580#,  6588
repl30 = $594c:  6595 ,  6598#
repl40 = $5954:  6600 ,  6604#
repl50 = $5962:  6612 ,  6615#
replc  = $5901:  3319 ,  3322 ,  6528#
replop = $5964:  6616#,  6622
res00  = $87aa: 17565 , 17568#
rescnt = $5fdb:  8053 ,  8152#
rese10 = $43eb:  2417#,  2421
rese20 = $43ec:  2418#,  2455
rese30 = $43f7:  2429#,  2433
rese40 = $43fe:  2430 ,  2438#
rese50 = $440b:  2446 ,  2452#
rese60 = $4412:  2424 ,  2460#
resend = $5fc6:  8114 ,  8133#
reser  = $43e2:  2278 ,  2292 ,  2306 ,  2406#
resetc = $5722:  6078 ,  6093#
resho  =   $28:   140#, 18006 , 18206$, 18260 , 18262$, 18267$, 18570
reslo  =   $2b:   143#, 15424$, 15456 , 18209$, 18248 , 18250$, 18270$, 18511$, 18579
reslst = $4417:  2304 ,  2305 ,  2473#,  4678 ,  4679
resmo  =   $2a:   142#, 15392$, 15416$, 15452 , 18208$, 18252 , 18254$, 18269$, 18576
resmoh =   $29:   141#, 18207$, 18256 , 18258$, 18268$, 18573
resnce =   $17:  1056#
resnum = $5fac:  8057 ,  8111#
ressnr = $5fd8:  8060 ,  8147#
resswp = $5fb7:  8056 ,  8064 ,  8120#
rest1  = $5adb:  6958 ,  6967#
rest4  = $5aeb:  6979 ,  6982#
rest5  = $5aef:  6985#
resto1 = $5ae1:  4898 ,  6948 ,  6974#
resto2 = $5ae6:  6969 ,  6978#
restor = $5aca:  2733 ,  6948#
resum1 = $5f8f:  8074 ,  8078 ,  8087#
resum2 = $5fa6:  8068 ,  8099 ,  8105#
resume = $5f62:  2758 ,  8047#
ret010 = $5270:  4952 ,  4962#
retpa1 = $a844: 24962 , 24966#
retpat = $a83b:  4977 ,  8380 , 24955#
retrn  = $976c: 21126 , 21164#, 21196
return = $5262:  2735 ,  4947#
rfat   = $a6db: 24553 , 24631#
rfata  = $a6e1: 24635 , 24641#
rgr    = $8182:  2864 , 15865#
rid    = $a6e5: 24544 , 24647#
rightd = $860a:  2861 , 17176#
rleft  = $85e3: 17138#, 17186
rleft1 = $85ea: 17138 , 17146#
rleft2 = $85eb: 17148#, 17217
rleft3 = $85ec: 17150#, 17221 , 17224
rlssrt = $5063:  4410 ,  4412#
rlsstk = $5059:  4405#,  4965 ,  6287 ,  8337
rmulc  = $8490: 16777 , 16778 , 16818#
rnd    = $8434:  2847 , 16743#
```

```
rnd0   = $8437: 16749#, 25537
rnd1   = $846a: 16749 , 16788#
rndd   = $96a9: 21024 , 21029 , 21034#
rndqnt = $aa68: 18890 , 25436#
rndrts = $8925: 17932 , 17947#
rndshf = $8917: 17937#, 18718
rndx   = $121b:   618#,  1736$, 16773 , 16774 , 16811 , 16812
rng100 = $5f0f:  7950 ,  7951 ,  7954 ,  7966#
rng200 = $5f26:  7970 ,  7981#
rngerr = $5f31:  7960 ,  7963 ,  7973 ,  7979 ,  7995#
rngrts = $5f30:  7982 ,  7990#
rnstln = $1170:   576#,  7015$,  7016$,  7039$,  7040$,  7088 ,  7089 ,  7581 ,  7584
rnum10 = $5b25:  7032 ,  7045#
rnum20 = $5b38:  7046 ,  7056 ,  7065#
rnumfg =   $77:   254#,  7178$,  7188$,  7361
rnuxit = $5bf8:  7193 ,  7234#
rolshf = $8990: 17801 , 18050#, 18878
romadd = $8a12: 16783 , 18132 , 18148 , 18172#, 19737 , 20534 , 20572
romdiv = $8a1e: 18136 , 18185#, 20682
rommlt = $8a08: 16779 , 18159#, 19220 , 19634 , 19723
romsub = $8a18: 18140 , 18179#, 20553 , 20694
romupk = $8a89: 10150 , 10171 , 18159 , 18172 , 18179 , 18185 , 18282#, 25539
rostln =   $5c:   191#,  7021$,  7022$,  7070$,  7073$,  7123 ,  7127 ,  7204 ,  7209
                 7544 ,  7549 ,  7594 ,  7597
rotang = $1158:   550#,  9563$
round  = $8c47:  5323 ,  7818 , 14014 , 18474 , 18632 , 18687 , 18706#, 25436 , 25533
rowcnt = $115f:   563#,  9057$,  9256 ,  9749$,  9764 ,  9772$,  9814 ,  9900 ,
                12962$, 13041 , 13060 , 13061 , 13062 , 13241 , 13246$, 13443
rows   = $dc01:  1436#
rpass1 = $5b68:  7025 ,  7066 ,  7109#
rpass2 = $5bae:  7114 ,  7132 ,  7138 ,  7177#
rreg   = $58bd:  2806 ,  6462#
rreg10 = $58e0:  6487 ,  6494#
rreglp = $58c1:  6468#,  6516
rrts   = $97fa: 21225 , 21229 , 21234 , 21237 , 21242 , 21263 , 21267#, 21283
rsca   = $a6d6: 24541 , 24625#
rsfn   = $a70d: 24556 , 24687#
rsmov  = $a714: 24692#, 24699
rspa   = $9676: 21000 , 21005#
rspcol = $8361:  2878 , 16439#
rsperr = $8379: 16443 , 16460#
rspmod = $835b: 16410 , 16429#
rspper = $83de: 16493 , 16509 , 16563#
rsppos = $8397:  2876 , 16490#
rspps1 = $83bd: 16515 , 16526#
rspps2 = $83c8: 16529 , 16532#
rspps3 = $83d5: 16540 , 16553#
rspr10 = $8354: 16408 , 16414 , 16418#
rspr20 = $8358: 16372 , 16391 , 16424#
rsprit = $831e:  2877 , 16369#
rstep  = $1172:   577#,  7018$,  7019$,  7050$,  7051$,  7055 ,  7618 ,  7622
rstvec = $0a00:   500#,  1683$, 23003 , 23007
rswplp = $5fb9:  8121#,  8128
rt     = $99c0: 21546 , 21549 , 21552 , 21555 , 21558 , 21561 , 21564#
rtts   = $9637: 20965 , 20969#
run    = $5a9b:  2731 ,  6911#, 25549
run10  = $5ab5:  6911 ,  6933#
run20  = $5abb:  6912 ,  6939#
```

```
runc   = $51f3:  4843#,  6925 ,  6934 , 19931 , 25550
runmod =   $7f:   270#,  3105 ,  3180 ,  3198 ,  3247 ,  3406 ,  3731 ,  3828$,  5126
                 6769 ,  6858 ,  6888$,  6917$,  8278 , 16893 , 16923 , 19886 , 19925
                22970 , 23608 , 23640 , 24873 , 24964$
runprg = $5aa6:  6924#, 25559
rusure = $a7e1:  6842 , 23595 , 23631 , 23816 , 24873#
rwnder = $8431: 16616 , 16650#
rwndo1 = $841e: 16619 , 16631#
rwndo2 = $8425: 16615 , 16637#
rwndo3 = $842d: 16626 , 16634 , 16640 , 16644#
rwndow = $8407:  2880 , 16607#
rwrt   = $a6f1: 24550 , 24657#
rwrt1  = $a6fa: 24658 , 24668#
sa     =   $b9:   320#
sabsz  = $1185:   591#, 25354 , 25355$, 25358 , 25359$
sadj2  = $5511:  5568 ,  5574#
sadj3  = $551e:  5578 ,  5583#
sadj4  = $5526:  5584 ,  5589#
sadj6  = $5535:  5599 ,  5605#
sadj8  = $5537:  5558 ,  5569 ,  5572 ,  5577 ,  5581 ,  5587 ,  5611#
sah    =   $ad:   318#, 12737
sal    =   $ac:   317#, 12734
sav13  = $877b:  8000 ,  8409 ,  9296 ,  9782 , 11271 , 17515#, 20067 , 24358
save   = $9112:  2741 , 19828#
saveit = $554a:  5631 ,  5636#
savenb = $911d: 19841#, 23511
savenp = $9115: 19833#, 23429
savin1 = $7794: 13667 , 13670#
savinp = $777c: 13547 , 13597 , 13648#
savquo = $8b79: 18493 , 18497 , 18501 , 18505#, 18527 , 18529
savram = $12b7:   692#, 12954$, 13113 , 13559$, 13564$, 13565$, 13568$, 13571$, 13573
                13574 , 24160$, 24692
savsiz = $03db:   479#,  9098$,  9102$,  9225$,  9226$,  9229 ,  9242$,  9245$,
                 9248$,  9250 ,  9253 ,  9278 ,  9281 , 13583$, 13584$, 13585$,
                13603 , 13606 ,  13610, 13637 , 13640
sbcflg = $1277:   647#,  1739$
sbcoll = $11e8:   596#
sbits  = $6cb3: 10970 , 10980 , 10988#, 11186 , 11191 , 11429 , 11443 , 12177 , 12257
                12264 , 12475 , 12884 , 13168 , 16412 , 16527 , 25029 , 25077
sc40_0 = $6a8b: 10417 , 10430#
sc40_1 = $6ab2: 10477 , 10484#
sc40_2 = $6ac8: 10467 , 10497 , 10501#
sc40_3 = $6ad0: 10503 , 10507#
sc40_4 = $6ad7: 10513#, 10515
scal10 = $696a: 10109 , 10116#
scal20 = $697d: 10128 , 10135#
scal30 = $6986: 10119 , 10146#
scal40 = $699e: 10155 , 10160#
scal50 = $69b9: 10176 , 10184#
scale  = $6960:  2777 , 10107#
scaleh = $7005: 11627 , 11844#
scalel = $6ff9: 11624 , 11828#
scalem = $116a:   570#,  1734$, 10116$, 10726$, 22376
scalen = $6ff2: 11591 , 11817#
scalex =   $87:   293#, 10135$, 10136$, 10188$, 10191$, 22381 , 22382
scaley =   $89:   294#, 10138$, 10139$, 10184$, 10185$, 22385 , 22386
scalxy = $9d4a:  9504 , 11104 , 11139 , 22376#, 22719 , 22760
scbot  =   $e4:   340#, 16623
```

```
schs1  = $9586: 20845 , 20849#
sclerr = $6a49: 10245 , 10264 , 10354#
sclf   =   $e6:   342#, 16633
sclr40 = $6a9b: 10431 , 10447 , 10456#
sclrts = $9d66: 22377 , 22399#
scncl1 = $6a8f: 10406 , 10436#
scncl2 = $6a92: 10445#, 10720
scnclr = $6a79:  2776 , 10406#
scnfil = $6b06: 10559#, 10593 , 10607 , 10620
scnflp = $6b0c: 10564#, 10566 , 10570
scnst  = $117f:   587#, 25038$, 25083$
scnthr = $5d83:  7230 ,  7598 ,  7608#,  7609
scratc = $a2a1:  2786 , 23628#
scrt   =   $e7:   343#, 16631
scrtch = $51d9:  1839 ,  4811 ,  4817#
sctc1  = $a2c1: 23648#, 23654
sctc2  = $a2ce: 23650 , 23657#
sctop  =   $e5:   341#, 16625
sddir1 = $1294:   670#, 25241 , 25260$, 25292$
sdfqh1 = $12a0:   674#, 25237 , 25239$, 25251 , 25268$, 25275 , 25300$, 25306
sdfql1 = $129d:   673#, 25233 , 25235$, 25249 , 25265$, 25281 , 25297$, 25303
sdmnh1 = $1291:   669#, 25252 , 25299
sdmnl1 = $128e:   668#, 25250 , 25296
sdmxh1 = $128b:   667#, 25267 , 25276
sdmxl1 = $1288:   666#, 12531$, 25264 , 25282
sdp1   = $a671: 24526#, 24580 , 24597 , 24636 , 24728
sdp2   = $a6a5: 24562 , 24570#
sdp3   = $a6ae: 24571 , 24579#
sdp4   = $a6b5: 24576 , 24589#
sdp5   = $a6b7: 24538 , 24567 , 24594#, 24626 , 24642 , 24652 , 24663 , 24672 , 24682
sdsph1 = $129a:   672#, 25238 , 25332 , 25335$
sdspl1 = $1297:   671#, 25234 , 25326 , 25330$
search = $4faa:  4233#,  4951 ,  6304 ,  7732 ,  8331
sendp  = $a667: 23190 , 23355 , 23423 , 23453 , 23494 , 23545 , 23830 , 24514#
ser    = $9553: 20803 , 20818#
setco1 = $7649: 13433 , 13438#
setcol = $763f: 13045 , 13358 , 13427#
setend = $5bad:  7167 ,  7172#
setenv = $6eb1: 11401 , 11493#
setexc = $5a81:  6770 ,  6887#,  6924 ,  6933 ,  6942 , 25560
setf10 = $6e74: 11426#, 11427
setf20 = $6e84: 11438 , 11445#
setf30 = $6e8f: 11458#, 11463
setfil = $6e6d: 11409 , 11418#
sethir = $9f4f: 10670 , 12770 , 22851#
setnxt = $5b9d:  7130 ,  7137 ,  7161#
setoct = $6ea8: 11398 , 11482#
setpcl = $6a5c:  1776 , 10348 , 10379#
setqut = $5716:  6069 ,  6083#
settwo = $9d8f: 21734 , 21839 , 21845 , 22415 , 22433 , 22453#
setvoi = $6e9d: 11395 , 11469#
setvol = $6edd: 11404 , 11523#
setwid = $71b6:  2825 , 12313#
sexp   = $9731: 21126#, 21147 , 21156
sftres = $4009:  1644 ,  1652#
sfur   = $9903: 21436#, 21448
sgn    = $8c65:  2840 , 18744#
sgnflg =   $69:   225#, 18942$, 19023 , 19712$, 19742$
```

```
sharp  = $122c:   628#, 11603 , 11680$, 11767$
shfarg = $8b89: 18522#, 18551
shftr2 = $8964: 18011#, 18030 , 18031
shftr3 = $8986: 18039#, 18056
shftr4 = $898c: 18040 , 18044#
shftrt = $899a: 18037 , 18058#
shiftr = $8979: 17736 , 18029#, 18865
shpn   = $96ee: 21006 , 21008 , 21080#
shwsp1 = $75d6: 13332#, 13373
shwsp2 = $75de: 13336#, 13367
shwsp3 = $75ee: 13344#, 13364
shwsp4 = $75f6: 13350 , 13355#
shwspr = $75d1: 12950 , 12994 , 13136 , 13313 , 13329#
sid    = $d400:   966#,  1857$,  1858$,  1859$,  1865$, 11460$, 11507$, 11510$,
                 11513$,  1516$, 11541$, 11749$, 11752$, 11758$, 11761$, 12359$,
                 12549$, 12556$, 12563$, 12574$, 16197 , 16201 , 25215$, 25304$,
                 25307$, 25321$
sigend = $4251:  1987 ,  1999#
sigmsg = $41bb:  1969 ,  1987 ,  1994#
sign   = $8c57:  7815 , 16743 , 18114 , 18723#, 18744 , 18797 , 25535
sign10 = $419d:  1969#,  1988
sign20 = $41aa:  1979#,  1982
sign30 = $41b2:  1971 ,  1984#
sign40 = $41b5:  1974 ,  1985#
signon = $419b:  1676 ,  1968#
signrt = $8c64: 18724 , 18736 , 18739#
sin    = $9410:  2851 , 20539#, 20591 , 25530
sin1   = $9442: 20558 , 20568#, 20613
sin2   = $9445: 20562 , 20570#
sin3   = $9452: 20575 , 20579#
sinco0 = $94ae: 20663#
sinco1 = $94a9: 20658#
sinco2 = $94a4: 20653#
sinco3 = $949f: 20648#
sinco4 = $949a: 20643#
sinco5 = $9495: 20638#
sincon = $9494: 20579 , 20580 , 20633#
sinval = $114a:   528#,  8899 ,  8918 , 11062$, 11064$, 21819$, 21822$, 21829
sizeok = $8731: 17444 , 17449#
sleep  = $6bd7:  2808 , 10775#
sleplp = $6bff: 10800#, 10803
slow   = $77c4:  2835 , 13726#
slpcnt = $0a1d:   502#, 10779$, 10780$, 10781$, 10789 , 10790 , 10791 , 10801 ,
                 10815$, 10816$, 10817$, 10826 , 10827$, 10830 , 10831$, 10834 ,
                 10835$
slr1   = $93f0: 20267 , 20440 , 20501#
slr2   = $9407: 20517 , 20520#
sltim2 = $6c0c: 10785 , 10810 , 10815#
sltim4 = $6c09: 10787 , 10792 , 10810#
snderr = $71f4: 12380#, 12402 , 12418 , 12502
sndth1 = $1285:   663#,  1885$, 12526 , 12587$, 25223 , 25314$, 25316
sndth2 = $1286:   664#,  1886$
sndth3 = $1287:   665#,  1887$
sndtl1 = $1282:   662#, 12590$, 25311 , 25315$
sner   = $a5b6: 24297 , 24309 , 24351#, 24412 , 24426 , 24444 , 24452
sner1  = $a2d4: 23592 , 23665#
sner2  = $a45a: 23933 , 24040#, 24101
sner3  = $a57d: 24197 , 24297#
```

```
snerr  = $796c:  2821 ,  3356 ,  3570 ,  4499 ,  5263 ,  5773 ,  6783 ,  6837 ,  7851
                 7995 ,  8147 ,  8244 ,  8450 , 11020 , 11214 , 13991 , 14231 ,
                14238#, 14556 , 14746 , 16364 , 17057 , 20021 , 20818 , 21474 ,
                22699 , 23163 , 23479 , 23665 , 23813 , 23993 , 24040 , 24351 , 24738
snerr1 = $4bab:  3282 ,  3290 ,  3332 ,  3356#,  3362 ,  3442
snerr4 = $55da:  5743 ,  5773#
snerr5 = $7881: 13869 , 13991#
snerr6 = $4c83:  3523 ,  3554 ,  3570#
sngflt = $84d4: 15707 , 15869 , 16027 , 16123 , 16309 , 16325 , 16419 , 16455 , 16480
                16521 , 16645 , 16887#, 17264 , 17287 , 21723 , 21897
sno    = $0132:   440#, 20921$, 20993 , 21049 , 21360 , 21363 , 21519 , 21523$
snvoic = $1281:   661#, 12383$, 12511 , 12541 , 12581
sord   = $1187:   592#
sound  = $71ec:  2762 , 12370#
sound1 = $71f7: 12376 , 12383#
sound2 = $721b: 12418#, 12473
sound3 = $7240: 12440 , 12455#
sound4 = $7261: 12494 , 12501#
sound5 = $7279: 12520#, 12521
sound6 = $727e: 12526#, 12527
sound7 = $7285: 12530#, 12539
sound8 = $72ad: 12566#, 12579
sound9 = $72b9: 12509 , 12581#
spdcmd = $748a: 13007 , 13016#
split  = $0a34:   503#, 10484 , 10709$
spmsb  = $11e6:   594#, 11179 , 11185 , 11193$, 12906 , 12907$, 16528 , 25076 ,
                25078$
spnumb = $12fc:   695#, 12873$, 12917 , 12938 , 12991 , 13167 , 13409
spos   = $998c: 21520 , 21525#, 21536
sprbas = $0e00:   505#, 12944 , 13302 , 13666
sprbit = $6c9b: 10888 , 10917 , 10929 , 10941 , 10953 , 10965#
sprbt1 = $6caf: 10976 , 10982#
sprcer = $71b3: 12285 , 12298 , 12308#
sprcl1 = $71a2: 12278 , 12293#
sprcl2 = $71b2: 12294 , 12303#
sprco1 = $6db1: 11204 , 11219#
sprco2 = $6dbc: 11210 , 11230#
sprco3 = $6dbd: 11207 , 11222 , 11225 , 11231#
sprcol = $7190:  2805 , 12277#
sprcor = $6d9e: 11010 , 11033 , 11201#
sprd05 = $738f: 12792#, 12796
sprd10 = $7397: 12801#, 12803
sprd15 = $73af: 12820#, 12829
sprd20 = $73b1: 12821#, 12825
sprd30 = $73c8: 12842#, 12847
sprd35 = $73d3: 12843 , 12850#
sprd40 = $73d6: 12855#, 12856 , 12876
sprd45 = $73e7: 12859 , 12871#
sprd50 = $7401: 12891 , 12895#
sprd55 = $7443: 12945 , 12948#
sprd60 = $744a: 12953#, 12956
sprd65 = $745a: 12968#, 12995 , 13011 , 13107 , 13171 , 13256 , 13321
sprdef = $7372:  2826 , 12770#
sprerr = $6c98: 10900 , 10960#, 10967 , 10998
sprit1 = $6c5c: 10880 , 10893#
sprit2 = $6c6f: 10894 , 10910#
sprit3 = $6c79: 10911 , 10922#
sprit4 = $6c83: 10923 , 10934#
```

```
sprit5 = $6c8d: 10935 , 10946#
sprit6 = $6c97: 10947 , 10955#
sprite = $6c4f:  2804 , 10875#
sproff = $6dd9:  1958 , 11052 , 11260#, 12918 , 16519 , 25034
sprs10 = $76f7: 13558#, 13561
sprs20 = $7720: 13548 , 13583#
sprs30 = $774b: 13610#, 13622
sprs40 = $775f: 13611 , 13624#
sprs50 = $7760: 13598 , 13627#
sprsav = $76ec:  2819 , 13547#
sptmp1 = $03e0:   481#, 13592$, 13627
sptmp2 = $03e1:   482#, 13595$, 13630
sqr    = $8fb7:  2846 , 19522#, 25524
sqr05  = $89b6: 18094#, 18130 , 18131
sqr20  = $89bb: 18099#, 18134 , 18135
squad  = $1180:   588#, 25047 , 25063
squeez = $8915: 17906 , 17932#
srch05 = $4faf:  4239#,  4296 ,  4298
srch10 = $4fbb:  4241 ,  4247#
srch15 = $4fdd:  4268 ,  4275#
srch20 = $4fe1:  4252 ,  4279#
srch30 = $4fef:  4276 ,  4285 ,  4292#
srch98 = $4ffb:  4245 ,  4300#
srch99 = $4ffd:  4257 ,  4263 ,  4273 ,  4281 ,  4301#
srchtk =   $02:    69#,  4233$,  4249 ,  4280
sreg   =   $05:    79#,  6442$,  6489
sscoll = $11e7:   595#, 16472 , 16477$, 25125 , 25126$
sshape = $642b:  2772 ,  9052#
sspeed = $117e:   586#,  1746$,  1825$,  1959$, 11055$, 11074$, 11083$, 12921$, 16520
                 25035
sswe   = $9679: 20991 , 20994 , 21006#
st2txt = $791f:  6104 , 14152#
stack  = $0100:   370#, 14694 , 15065 , 15068 , 15072$, 15075$
stasgn = $4cf6:  3648 ,  3651 ,  3655#
stash  = $aa1f:  2828 , 25385#
stashx = $02b9:   454#, 15717$
status =   $90:   300#
stenv1 = $6eb2:  1898 , 11498#
sterr  = $77ea: 13791#, 13824
sth010 = $9f54: 22856 , 22862#
sth020 = $9f6d: 22869 , 22873 , 22882#
sth030 = $9f92: 22923#, 22947
sth040 = $9f9a: 22924 , 22932#
sth050 = $9fa0: 22933 , 22935#
sth060 = $9fa8: 22940 , 22942#
sth070 = $9fb2: 22927 , 22950#
sth080 = $9fc7: 22967#, 23103
sth090 = $9fd9: 22975 , 22982#
sth100 = $9ff2: 23002#, 23038 , 23040
sth110 = $9ffe: 22971 , 23010#
sth120 = $9fff: 23004 , 23008 , 23015#
sth130 = $a010: 23019 , 23030#
sth140 = $a017: 23027 , 23035#
stkbot = $09ff:   496#,  4240 ,  4244 ,  4872 ,  4873
stkini = $5238:  1656 ,  3802 ,  3967 ,  4903#
stkrts = $524f:  4850 ,  4924#
stktop = $0800:   486#,  1816$,  4315
stline = $766e: 12842 , 13471#
```

```
stmdsp = $46fc:  2721#,  3308 ,  3311 ,  3340
stnol1 = $6f11: 11571#, 11579
stnol2 = $6f1d: 11572 , 11582#
stnoln = $6f07: 11324 , 11565#
stoint = $84e5: 14182 , 16874 , 16910#
stold  = $9043: 19638 , 19641#
stolop = $4f06:  4095#,  4099
stomlt = $7d65: 15198 , 15203#
stop   = $4bcb:  2737 ,  3395#
stop1  = $4bd0:  3379 ,  3406#
stopc  = $4bce:  3395 ,  3401#
stopc0 = $4bc1:  3384#,  3385
stopc1 = $4bbb:  3370 ,  3377#
stopnb =   $8b:   295#,  8608$, 22114
storts = $84ef: 16894 , 16918#
stout  = $96d8: 21063#, 21064
stp    = $95f8: 20926#, 20940
stpsp1 = $4191:  1958#,  1962
stpspr = $418d:  1654 ,  1955#
str1   =   $5d:   194#,  6582$,  6611 ,  6618$, 20275$, 20279$, 20444$, 20449$,
                 20503$, 20507$, 20514 , 20515$, 20518$, 21629$, 21658 , 21676
str2   =   $60:   195#,   204#,  6585$,  6591 ,  6593$,  6596$,  6599 ,  6602 ,  6616
                 21635$, 21642 , 21654 , 21666 , 21683
stradj = $54f6:  5503 ,  5526 ,  5555#, 17537
strcmp = $4cce:  3613 ,  3629#
strcnt = $1160:   565#,  9903$,  9905 ,  9906$, 13330$, 13339 , 13341$
strd   = $85ae:  2856 , 17089#
strdn2 = $5744:  6106 ,  6120#
strdon = $5554:  5646#,  5682
strend =   $33:   148#,  4884$,  4885$,  8632 ,  8635 ,  8715 ,  8719 , 14773 , 14774
                14808 , 14811 , 14960$, 14961$, 15108 , 15111 , 15261$, 15262$, 15285
                15291 , 15554 , 15557 , 20200 , 20204
strfi1 = $86bc: 17332 , 17344#
strfi2 = $86bd: 17342 , 17345#
strfin = $86b8: 17335 , 17341#
strfre = $92c0: 20202 , 20210#
strget = $86aa: 17330#, 17338
strini = $8688:  5479 , 14329 , 17297#, 17361 , 17449
strlit = $869a:  5687 ,  5782 , 14147 , 17097 , 17316#
strlt1 = $86cc: 13580 , 17361#
strlt2 = $86a0:  6103 , 17323#
strna1 = $7b0b: 14629 , 14634#
strnam = $7aff: 14595 , 14624#
strnex = $847a: 16768 , 16800#
strng1 =   $70:   239#,  2163 ,  2169 ,  5484$,  5485$,  5489 ,  5490 , 13576$,
                13577$, 17323$, 17324$, 17348 , 17351 , 17405$, 17432$, 17435$,
                17457 , 17458
strng2 =   $72:   240#, 17349$, 17354$
strout = $55e2:  5782#, 19178
strpr2 = $55ec:  5792#,  5799 ,  5802
strprt = $55e5:  5646 ,  5688 ,  5787#,  5916
strptr = $1155:   544#,  9112$,  9181$,  9183 ,  9344$,  9355$,  9357
strrt1 = $79b3: 14274 , 14280 , 14284 , 14302#, 14308 , 14311
strrts = $7a27: 14355 , 14415#
strspa = $8690:  9121 , 15755 , 15819 , 17107 , 17151 , 17306#, 20879
strst2 = $86c9: 17352 , 17354#
strst4 = $86d3: 17367#, 17374
strsz  = $1153:   537#,  9300$,  9323
```

```
strtx2 = $791c: 14144 , 14147#
strtxt = $7913:  5911 , 14141#
stxbuf = $8f17: 19353 , 19359#
stxfnd = $7b13: 14643#, 14676
stxtpt = $5254:  4102 ,  4843 ,  4934#,  7239 ,  7587 , 19937
stzl   = $97a8: 21208#, 21210
subit  = $888e: 17808 , 17815#
sublfg =   $12:   115#,  4921$,  7725$, 14599 , 14627 , 14635$, 16942$, 16974$
subtw2 = $9d7f: 22405 , 22438#
subtwo = $9d7c:  9547 ,  9608 , 21956 , 21978 , 21991 , 22410 , 22433#, 22464
susmus = $701b: 11874#
susrel =   $06:  1011#, 11510$, 12563$
sustab = $1249:   640#, 11509 , 12086 , 12222$
swap   = $aa29:  2832 , 25397#
swap01 = $aa2b: 25386 , 25392 , 25398#
swaper = $aa65: 25420 , 25431#
swaplp = $9061: 19664#, 19671
swapxy = $8139: 11098 , 15769 , 15775 , 15791#
swe    = $0127:   429#, 21005$, 21044$, 21075$, 21208$, 21400 , 21403$
sxswt  = $1181:   589#, 25353 , 25357
synchr = $795e:  5049 ,  5298 ,  5634 ,  5856 ,  5883 ,  5914 ,  6572 ,  7772 ,
                14227#,  16949 , 16972 , 20822 , 23671
syntmp =   $79:   262#,  3678$,  3681 ,  4434$,  4437 ,  4447$,  4450 ,  8498$,  8511
                 8555 , 14228$, 14230 , 15169$, 15172 , 15328$, 15333 , 15340$, 15341
                17129$, 17132 , 17180$, 17184 , 17212$, 17216 , 17439$, 17443 ,
                19094$, 19105 , 21678$, 21686
sys    = $5885:  2751 ,  6415#
sys10  = $589c:  6429 ,  6432#
sys20  = $58a3:  6433 ,  6436#
sys30  = $58aa:  6437 ,  6440#
sys40  = $58b1:  6441 ,  6444#
syswt  = $1183:   590#
szer   = $9816: 21271 , 21286#
t3     = $1174:   578#, 11091$, 23256$, 23270$
t4     = $1175:   579#, 23259$, 23273
t5     = $1176:   580#, 23265$, 23274
taber  = $55b9:  5667 ,  5671 ,  5736#
tabld  = $a627: 23166 , 23330 , 23351 , 23421 , 23451 , 23492 , 23543 , 23598 , 23634
                23706 , 23738 , 23774 , 23785 , 23797 , 23824 , 24459#, 24537
tacky1 = $a493: 24075 , 24091#
tag1   = $974d: 21146#, 21148
tag2   = $9758: 21141 , 21144 , 21153#, 21157
tag3   = $9782: 21146 , 21182#, 21248
tan    = $9459:  2852 , 20586#, 25531
tansgn =   $14:   125#,  3691 , 14049$, 20564 , 20566$, 20589$, 20604
tat    = $98da: 21412#, 21419
tdossa = $11ed:   601#, 23697$, 24625
telx   = $98a1: 21357 , 21373#
temper = $6fe1: 11780 , 11788#
tempf1 =   $59:   182#,  4340 ,  4345 ,  4351 , 18626 , 18627 , 19698 , 19702 , 19703
                20597 , 20598 , 22440$, 22446
tempf2 =   $5e:   183#, 18620 , 18932$, 19739 , 19740
tempf3 =   $50:   169#, 20593 , 20594 , 20607 , 20608
tempo  = $6fd7:  2802 , 11774#
temppt =   $18:   134#,  1781$,  4904$, 17382 , 17415$, 17630$, 20259 , 20435
tempst =   $1b:   136#,  1780 ,  4903 ,  8641$,  8645$,  8755 ,  8760 , 17401
tenc   = $8b2e: 18438#, 18451 , 18452
```

```
tenexp =   $60:   197#, 14289$, 18995 , 19005 , 19008$, 19014$, 19020$, 19069 , 19088
                19108$, 19272$, 19389 , 19395
testwd = $5af0:  6996#,  7284 ,  7285
timco1 = $8fa3: 19496#
timco2 = $8fa7: 19501#
t               timco3 = $8fab: 19506#
timco4 = $8faf: 19511#
timco5 = $8fb3: 19516#
timcon = $8f9f: 14297 , 19491#
timelp = $5418:  5373#,  5398
timend = $8fb7: 19371 , 19517#
times2 = $6fe4: 11421 , 11699 , 11793#
timnum = $5448:  5377 ,  5384 ,  5411#
timstr = $85b8: 14299 , 17095#
tmdes1 = $03d6:   476#, 21576$, 21579$, 21619 , 21717 , 21718
tmdes2 = $03d8:   477#, 21587$, 21590$, 21712 , 21713
tmerr  = $5757:  6132 ,  6140#
tmfrqh = $12ad:   685#, 12393$
tmfrql = $12ac:   684#, 12392$, 12566
tmmaxh = $12a6:   678#, 12390$
tmmaxl = $12a5:   677#, 12389$, 12530 , 12538
tmminh = $12a8:   680#, 12433$
tmminl = $12a7:   679#, 12432$
tmpdir = $12a9:   681#, 12421$
tmplin = $1216:   616#,  8167$,  8221 ,  8225 ,  8342 ,  8343
tmpot1 = $12b1:   689#, 16175$, 16208 , 16274$, 16284$, 16286
tmpot2 = $12b2:   690#, 16204$, 16244
tmprat = $1222:   623#,  1854$, 11782$, 25194
tmptk  =   $77:   252#,  5838$,  6007
tmptrp = $120d:   611#,  3754$,  8139
tmptxt = $1214:   615#,  8164$,  8213 ,  8217
tmpul  = $12ae:   686#, 12504$, 12538
tmpulh = $12af:   687#, 12505$
tmstph = $12ab:   683#, 12455$
tmstpl = $12aa:   682#, 12458$
tmtimh = $12a4:   676#, 12405$, 12508 , 12584
tmtiml = $12a3:   675#, 12404$, 12507 , 12583
tmwave = $12b0:   688#, 12486$
tod10  = $dc08:  1469#
todhr  = $dc0b:  1490#
todmin = $dc0a:  1483#
todsec = $dc09:  1476#
togcur = $764a: 12968 , 12977 , 13443#
ton110 = $714e: 12194 , 12205 , 12217#
tone10 = $70cb: 12072 , 12081#
tone20 = $70e2: 12095#, 12158
tone30 = $7100: 12113 , 12135#
tone40 = $7119: 12136 , 12152#
tone80 = $7133: 12174 , 12188#
tone90 = $7136: 12163 , 12193#
tonerr = $70c8: 12076#, 12175
tonnum = $123a:   636#, 12081$, 12211 , 12217
tonval = $123b:   637#, 12084$, 12087$, 12090$, 12127 , 12130$, 12147 , 12150$,
                12188$, 12200 , 12218 , 12221 , 12224
```

```
tos    = $7d:   269#,  4308 ,  4309$,  4311 ,  4314$,  4319 ,  4383 ,  4386 ,
                4395$,  4398$,  4407 ,  4408$,  4411$,  4875$,  4876$,  6795$,
                6799$,  6803$,  6807$,  6811$,  7754$,  7759$,  7765$,  7769$,
                7789$,  7825$,  7834$,  7838$,  7844$,  8214$,  8218$,  8222$,
                8226$,  8230$
tranr  = $a6bf: 24529 , 24607#
trans  = $a397: 23605 , 23636 , 23708 , 23727 , 23750 , 23776 , 23787 , 23799 ,
                23830#
trap   = $5f4d:  2759 ,  8026#
trap1  = $5f5c:  8029 ,  8040#
trapno = $120b:   610#,  3377 ,  3748 ,  3756 ,  3760$,  4861$,  8034$,  8041$,
                8140$
trcflg = $116f:   575#,  1728$,  3244 ,  4827$,  6456$
tripfg = $1276:   646#,  1738$,  3117 ,  3121$,  6899$, 25154$
trmlp  = $5786:  6175#,  6182
trmno1 = $575b:  6131 ,  6143#
trmok  = $5784:  6121 ,  6124 ,  6173#
trmpos =   $0b:    93#,  5739$,  5749
troff  = $58b7:  2761 ,  6455#
tron   = $58b4:  2760 ,  6449#
trts   = $999a: 21452 , 21532#
tryag2 = $929b: 20173#, 20249
tryag3 = $92a9: 20185 , 20188#
tryag4 = $92b6: 20197 , 20200#
tryof1 = $8b04: 18388 , 18393#
tryoff = $8af9: 18376 , 18383#
tstdun = $510a:  4567 ,  4575#
tstgo  = $629a:  8745 ,  8746 ,  8768#
tstop  = $7809: 13837#, 17461
tstval = $627c:  8681 ,  8692 ,  8742#
tstxit = $629d:  8749 ,  8770#
tto    = $4b34:  3185 ,  3231#,  3409
turnon = $7af8: 14589 , 14618#
twoby1 = $9dca: 22513#, 22532
twoby2 = $9ddf: 22515 , 22528#
twobyt = $9dae: 21849 , 22391 , 22495#
twopi  = $948a: 20541 , 20542 , 20623#
tworts = $9dee: 22536 , 22539#
txtptr =   $3d:   153#,  2035$,  2037$,  2042 ,  2109 ,  2204 ,  2207 ,  2315$,  2338
                2341$,  2344$,  2350 ,  2375$,  2379$,  2389 ,  2392 ,  2399$,  2418
                3131 ,  3134 ,  3157$,  3160$,  3217 ,  3218$,  3221$,  3231 ,  3232
                3841$,  3842$,  3971 ,  4095 ,  4937$,  4941$,  4969$,  4973$,  4994
                4995$,  4998$,  5149 ,  5150$,  5153$,  5210$,  5212$,  5987 ,
                5991$,  6052$,  6053$,  6074$,  6096 ,  6097 ,  6165$,  6166$,
                6175 ,  6179$,  6378$,  6382$,  6739 ,  6741 ,  6763$,  6767$,
                6794 ,  6798 ,  6875$,  6876$,  7169$,  7170$,  7216$,  7220$,
                7330 ,  7333 ,  7347$,  7350$,  7354 ,  7356$,  7357$,  7479$,
                7636$,  7638$,  7644 ,  7647 ,  7753 ,  7756 ,  8097 ,  8098$,
                8100$,  8125$,  8163 ,  8299 ,  8300$,  8303$,  8372$,  8376$,
                13591 , 13594 , 13628$, 13631$, 13797 , 13799$, 13800$, 13950 ,
                13952$, 13953$, 14141 , 14142 , 14155$, 14156$, 16959 , 16962
                17025 , 17028 , 17032$, 17036$, 17061$, 17064$, 22984 , 22985$
txttab =   $2d:   145#,  1788$,  1789$,  4150 ,  4151 ,  4417 ,  4418 ,  4822$,
                4825$,  4831 ,  4836 ,  4935 ,  4939 ,  6750 ,  6751 ,  6975 ,
                6976 , 19836 , 19868 , 19869 , 22956 , 22958$, 23091 , 23093$
```

```
txttop = $1210:  613#,  3883 ,  3884 ,  3910 ,  3923 ,  3924$,  3927 ,  3929$,  3936
                3981 ,  3982 ,  4011$,  4012$,  4204$,  4205$,  4834$,  4838$,  7180
                7181 ,  7469$,  7471$,  7499 ,  7501$,  7502$,  7650 ,  7653 ,  7906
                7907$,  7910 ,  7911$,  7926 ,  7940$,  7943$, 15511 , 15514 , 19833
               19834 , 19922$, 19923$, 22862 , 22888 , 22896 , 22899 , 22952 ,
               22954$, 23084 , 23095 , 23097$
typlin = $510c:  4570 ,  4580#
uexp   = $0129:  431#, 20962$, 20978 , 20985$, 21012 , 21132 , 21137 , 21337 , 21352
umlcnt = $7e6c: 15446 , 15461#
umult  = $7e3e: 15238 , 15361 , 15413#
umultc = $7e53: 15432#, 15462
umultd = $7e49: 15395 , 15424#
unadju = $81f3: 16022 , 16032#
unit   = $a58d: 24013 , 24259 , 24314#
unit1  = $a43d: 23974 , 24013#
unit2  = $a553: 24227 , 24259#, 24290
unpgp1 = $819a: 15901 , 15916#
unpgpm = $818c: 10408 , 15866 , 15874#
unpstk = $78b8: 14041 , 14045#
unqu10 = $537e:  5210#,  5217
unqu20 = $5384:  5211 ,  5213#
unquot = $537c:  5162 ,  5205#,  8269
unqurt = $5390:  5214 ,  5222#
upgrap = $11ec:  600#,  1834$,  9857 ,  9928
upplow = $11eb:  599#,  1837$,  9919
uround = $97b9: 21007 , 21034 , 21222#
userr  = $5a15:  6757 ,  6777#,  6962
usgn   = $0128:  430#, 20960$, 21020 , 21143 , 21169 , 21171$, 21346
using  = $9520:  5661 , 20776#
usrpok = $1218:  617#,  1695$,  1700$,  1701$,  2843
val    = $804a:  2857 , 15564#
val1   = $8052: 15565 , 15574#, 25511
val2   = $8075: 15539 , 15603#
valtyp =  $0f:  105#,  3630$,  5303 ,  5681 ,  6060 ,  6480 ,  9060 , 13650 , 13771
               13892 , 13942$, 14087$, 14260 , 14560$, 14588$, 15036 , 15089$,
               16911$, 17272$, 17408$, 20829
varnam =  $47:  158#, 14257 , 14258 , 14546$, 14610 , 14611$, 14624$, 14657 , 14662
               14749 , 14750 , 14913 , 14919 , 15047 , 15049 , 15055$, 15057$, 15120
               15124 , 15188 , 15196 , 15381 , 15386
varok  = $7b90: 14715 , 14725 , 14732 , 14742 , 14767#
varpnt =  $49:  159#, 14944$, 14945$, 15399$, 15403$, 15406 , 16953 , 16956 ,
               17004$, 17011$, 17014 , 17020 , 17038 , 17041 , 19343$, 19361
vartab =  $2f:  146#,  1794$,  1795$,  4878 ,  4879 ,  5440 ,  5445 , 14637 , 14638
vartxt =  $4d:  162#,  5988$,  6178
vdcadr = $d600:  1263#, 10341$, 16012$, 16302$, 16314
vdcchr = $2000:  1261#
vdcdat = $d601:  1286#, 10343 , 10346$, 16014 , 16304
vdcscn =  $00:  1260#
verck  =  $93:  312#
verify = $9129:  2742 , 19851#
vf     = $012c:  434#, 20971 , 20996 , 20999$, 21040 , 21081 , 21483$, 21495$
vic    = $d000:  832#, 10975 , 10982$, 13166 , 13169$, 16413 , 25013$
vicbkg = $d021:  915#,  9896 , 10276$, 10390 , 12812 , 13404 , 13526 , 15960
vicbrd = $d020:  910#, 10308$, 15989
viccr1 = $d011:  844#, 13707 , 13713$, 13735 , 13742$
vicifr = $d019:  881#, 25092 , 25093$
viclpx = $d013:  857#, 25165
viclpy = $d014:  861#, 25168
```

```
vicmc0 = $d025:   929#, 12288$, 13393 , 16446
vicmc1 = $d026:   934#, 12301$, 13398
vicmcs = $d01c:   898#, 12887
vicmoe = $d015:   865#, 12915$, 13125$, 25025
vicspc = $d027:   939#, 10905$, 12992$, 13410 , 16397
vicspd = $d030:   952#, 13720$, 13733$
vicssc = $d01e:   904#, 25117 , 25135$
vn     = $012a:   432#, 20952$, 21038$, 21041 , 21059 , 21072 , 21082 , 21096$,
                 21111$, 21266$
voice  = $122f:   630#,  1895$,  1900$,  1903$, 11420 , 11476$, 11499 , 11698
voichi = $1224:   625#,  1888$,  1889$,  1890$, 11426 , 11704 , 11711$, 11733 ,
                 11734$, 12520 , 25189 , 25199 , 25201$
voiclo = $1223:   624#, 11660 , 11708$, 11729 , 11730$, 25192 , 25195$
volerr = $71e9: 12338 , 12365#
voltab = $703c: 11534 , 11539 , 11921#
volume = $71c5:  2763 , 12333#
vtemp1 =   $8e:   297#,  9078$,  9086 ,  9107$,  9108$,  9152$,  9180 , 13074$, 13080
                 13081$, 13087 , 13300$, 13308 , 13407$, 13412 , 13415$, 13421 ,
                 21781$, 21794$, 22495$, 22513$
vtemp2 =   $8f:   298#,  9172$,  9210 , 13034$, 13085 , 13092$, 13305$, 22496$,
                 22514$
vtemp3 = $1177:   581#, 22508$, 22523 , 22525$, 22528$, 22535
vtemp4 = $1178:   582#, 22687$, 22712 , 22725 , 22754$, 22755$, 22757 , 22763 , 22780
                 22805
vtemp5 = $1179:   583#, 22722$, 22733$, 22744$, 22768$, 22801$
vwork  = $1131:   510#,  8675 ,  8678$,  8685 ,  8687$,  8695 ,  8698$,  8729$,  8773
                  8841$,  8871 ,  8880$,  8893 ,  8899 ,  8906$,  8918 ,  8922 ,  8926
                  8935$,  8949 ,  8952 ,  8953 ,  8972$,  8975$,  8977 ,  8981 ,  9030
                  9033$,  9034$,  9037$,  9039$,  9068 ,  9071 ,  9074 ,  9075 ,
                  9091$,  9094$,  9104 ,  9105 ,  9129 ,  9161 ,  9162$,  9273$,
                  9420$,  9448$,  9485 ,  9503 ,  9545 ,  9546 ,  9556$,  9558$,
                  9606 ,  9607 ,  9616 ,  9623 ,  9643 ,  9644 ,  9662 ,  9667 ,
                  9713 , 11103 , 11106 , 11138 , 21829 , 21832 , 21860 , 21863$,
                 21870 , 21905 , 21913 , 21914 , 21950 , 21951 , 21951 , 21954 ,
                 21955 , 21980$, 22037 , 22039$, 22054$, 22059 , 22063$, 22296 ,
                 22340 , 22392$, 22396$, 22421 , 22439 , 22453 , 22498 , 22503$,
                 22520 , 22554$, 22669 , 22670$, 22713 , 22727$, 22738$, 22774$
                 22809$
wait   = $6c2d:  2739 , 10842#
wait10 = $6c3c: 10847 , 10850#
waitlp = $6c3e: 10855#, 10862
warmvc = $4003:  1644#,  1682
wave   = $1230:   631#, 11502$, 11760 , 25208
wavmus = $7025: 11888#
wavtab = $1253:   641#, 11501 , 12089 , 12225$
widerr = $71c2: 12320 , 12328#
width  = $116b:   571#,  1733$, 12322$, 22051
window = $72cc:  2823 , 12598#
wintmp = $12b3:   691#, 12611$, 12622$, 12639$, 12640 , 12652$, 12653 , 12669 , 12670
                 12674 , 12675
wnderr = $7332: 12609 , 12620 , 12637 , 12641 , 12650 , 12654 , 12664 , 12691#
wndw10 = $72d7: 12605 , 12609#
wndw20 = $72f1: 12633 , 12637#
wndw30 = $732e: 12682 , 12685#
xabs   = $1139:   519#, 21955
xayslp = $6c16: 10786 , 10794 , 10823#
xcircl = $1150:   533#,  9485 ,  9643
xcnt   = $0110:   375#, 24514$, 24528$
```

```
xcord1 = $1150:   532#,  8952
xcord2 = $115c:   555#,   556#,  8840 ,  8953 ,  9219$,  9527$,  9528$,  9546 ,  9600
                 9601$,  9604$,  9607 ,  9623
xcount = $1156:   545#,  8866$,  9366$,  9386 ,  9392 ,  9402
xdest  = $1135:   515#,  8959$,  8964$,  9071 ,  9105 ,  9447 ,  9672$,  9713 ,
                 11025$, 11028$, 11103 , 11106 , 11114$, 11116$, 11138 , 11167 ,
                 11177$, 11188$, 21950 , 22545 , 22545 , 22548 , 22553 , 22556
xdist1 = $1154:   539#, 21860
xeqchr = $4b91:  3325#,  3352
xeqcm  = $4a9f:  3100#,  3223 ,  3364
xeqcm1 = $4b3f:  3173 ,  3242#,  5112
xeqcm2 = $4b59:  3245 ,  3248 ,  3267#,  5270
xeqcm3 = $4b64:  3271 ,  3278#
xeqcm4 = $4b76:  3287 ,  3297#
xeqcm5 = $4b7e:  3299 ,  3306#,  3341
xeqdir = $4af3:  3173#,  3852
xeqes1 = $4ba5:  3335 ,  3338 ,  3346#
xeqesc = $4b94:  3268 ,  3331#
xeqmid = $4b8b:  3279 ,  3319#
xeqrts = $4b3e:  3237#,  3242
xor    = $83e1:  2879 , 16568#
xradus = $1154:   542#,  9491$,  9492$,  9503 ,  9511 ,  9514 ,  9562 ,  9572 ,
                 9629$,  9644 ,  9667
xrcos  = $1160:   564#,  8858 ,  9058$,  9259 ,  9573$,  9628
xreg   =   $07:    81#,  6434$
xsave  = $115d:   559#,  9115$,  9118$,  9272 ,  9348$,  9351$,  9425 ,  9434$,
                 9445$
xsize  = $1159:   551#,  9068 ,  9090 ,  9093 ,  9097$,  9101$,  9104 ,  9198 ,
                 9208$,  9279$,  9340$,  9347 ,  9350 ,  9444
xspac  = $55d0:  5746 ,  5760#
xspac1 = $55dd:  5762 ,  5776#
xspac2 = $55d1:  5761#,  5777
xtsner = $6041:  8179 ,  8244#,  8318
xyabs  = $1139:   518#,  8960$,  8965$, 21936$, 21942$, 22000 , 22002$, 22005 ,
                22007$
xypos  = $1131:   511#,  8657 ,  8659$,  8660$,  8666$,  8668$,  8677$,  8689$,
                 8697$,  8700$,  8702$,  8908$,  8937$,  9114 ,  9117 ,  9165$,
                 9268$,  9270$,  9276$,  9422$,  9453$,  9455$, 10513$, 21866$,
                21907 , 21982$, 22057$, 22062$, 22292 , 22334 , 22353 , 22358 ,
                22425 , 22443 , 22456 , 22499 , 22505$, 22524 , 22557$, 22714$,
                22716 , 22717$, 22730$, 22740$, 22777$, 22806$
xysgn  = $113d:   522#,  9799$,  9821 ,  9842 , 10012 , 12784$, 12818$, 21923$,
                21924$, 21935$, 22038
yabs   = $113b:   521#, 21954
ycircl = $1152:   536#,  9662
ycord1 = $1152:   535#,  8981
ycord2 = $115e:   560#,  8977 ,  9275 ,  9436 ,  9438$,  9533$,  9534$,  9545 ,  9606
                 9616
ycount = $1158:   549#,  8926
ydest  = $1137:   516#,  9075 , 11026$, 11029$, 11135$, 11136$, 11147 , 21914 , 22548
ydist2 = $115a:   552#,  8859$,  8885 ,  8890$,  9213$,  9282$
ypos   = $1133:   513#, 21913
yradus = $1156:   546#,  9497$,  9498$,  9512$,  9519$,  9522$
yreg   =   $08:    82#,  6438$
ysgn   = $113f:   523#, 21917$, 21918$
ysize  = $115b:   553#,  8862$,  8887$,  9074 ,  9216 ,  9266$,  9458 ,  9460$,  9462
                 9464$
zeremv = $8b0f: 18372 , 18383 , 18403#
```

```
zerita = $7db0: 15272#, 15274 , 15278
zero   = $03d2:   474#,  5357 , 14279 , 14283 , 14382 , 14386 , 14704 , 14705 , 16351
zerof1 = $88d8: 17878#, 19536
zerofc = $88d6: 15569 , 17876#, 17923 , 18405
zeroml = $88da: 17880#, 18390
zerot  = $9898: 21367#, 21401
zerot1 = $98b4: 21368 , 21388#
zout   = $98a6: 21371 , 21377#, 21390
zxit   = $a2d3: 23632 , 23641 , 23660#
_aryhd =   $05:   174#, 14849 , 15471
_aryln =   $03:   175#, 14900
_beats = $0480:    55#,  1844 ,  1845 , 11565 , 11568
_begnt =   $18:  2673#,  5065 ,  5105 ,  5187 ,  5235
_bendt =   $19:  2675#,  5182
_blink =   $02:    15#,  6701
_bnk   =   $01:   284#, 12703 , 23465 , 23522 , 24332 , 24341
_bufln =   $a1:   452#,  4223
_cass  =   $01:   129#, 19971 , 20037
_cleft =   $9d:    37#, 13477
_clr   =   $93:    32#,  1994 , 10528 , 10547 , 12684 , 13477
_clrt  =   $9c:  2513#, 10635
_cmdsa =   $0f:   401#, 23703 , 23867 , 24823
_col40 =   $28:   346#,  9736 ,  9939
_col80 =   $50:   347#
_colit =   $17:  2671#,  7315
_cr    =   $0d:    19#,  1994 ,  1995 ,  1996 ,  1997 ,  1998 ,  3425 ,  3702 ,  3702
                 4214 ,  5706 ,  5798 ,  6160 ,  6225 ,  8580 ,  9017 ,  9018 ,  9019
                12858 , 13288 , 13478 , 19891 , 19891 , 23298 , 23643 , 23657 , 24839
                24884
_crite =   $1d:    36#,  4141 ,  5819 , 13477
_crsdn =   $11:    25#, 13477
_crsup =   $91:    26#, 13477
_d1    =   $10:   277#, 12702 , 23158 , 23170 , 23416 , 23443 , 23464 , 23521 , 23744
                23756 , 23757 , 23763 , 23792 , 23804 , 23807 , 23808 , 24080 , 24090
_d2    =   $20:   278#, 23324 , 23345 , 23756 , 23757 , 23763 , 23804 , 23807 , 23808
                24238 , 24247 , 24733 , 24754
_datat =   $83:  2320 ,  2480#,  6262
_deffa =   $08:   397#, 23866 , 24819
_defla =   $00:   394#, 23198 , 23217 , 23317 , 23674 , 23700 , 23865 , 24822 , 24830
                24854
_deflt =   $0a:    46#,  7012 ,  7013
_defsa =   $00:   400#, 23196 , 23448
_dot   =   $eb:  2617#,  8229 ,  8258 ,  8330
_drv1  =   $d1:   381#, 24464 , 24468 , 24472 , 24481 , 24485 , 24489 , 24493 , 24497
                24501 , 24501 , 24505 , 24570
_drv2  =   $d2:   387#, 24476 , 24477 , 24493 , 24497 , 24579
_dscln =   $07:   173#, 14673 , 14780 , 14889 , 14928
_dsdln =   $28:   267#, 24795 , 24804 , 24844 , 24909
_ecmdt =   $fe:  2368 ,  2643#,  3267 ,  4669 ,  4768 ,  5060 ,  5099 ,  5178 ,  5227
                 7309 ,  8441
_efunt =   $ce:  2373 ,  2582#,  3438 ,  4672
_elset =   $d5:  2592#,  3289 ,  3292 ,  3441 ,  5080 ,  7003
_endad =   $04:   286#, 23465 , 23473 , 23474 , 24139 , 24147
_endt  =   $80:  2474#
_eom   =   $00:    14#,  1998 ,  2235 ,  2642 ,  2691 ,  2709 ,  3019 ,  3425 ,  3702
                 3805 ,  5695 ,  6076 ,  6160 ,  6225 , 13466 , 13471 , 14325 , 19418
                19430 , 19891 , 24850 , 24878
_eqult =   $b2:  2547#,  5297 ,  6571 , 16948
```

```
_erbd  =   $18:  3000#,  6149
_erbdc =   $28:  2925 ,  3048#
_erbdk =   $24:  3036#, 23619
_erbrk =   $1e:  3018#,  3389 , 19762
_erbs  =   $12:  2982#, 15141
_ercn  =   $1a:  3006#,  6863
_ercr  =   $1f:  3021#,  8152
_erdd  =   $13:  2985#, 15156
_erdnp =   $05:  2943#
_erdv0 =   $14:  2988#, 18443
_erfc  =   $0e:  2970#, 15147
_erfnf =   $04:  2940#, 23718
_erfno =   $03:  2937#
_erfo  =   $02:  2934#
_erid  =   $15:  2991#, 16898
_eridn =   $09:  2955#, 24399
_erlnf =   $20:  3024#,  8348
_erlod =   $1d:  3015#, 19913
_erls  =   $17:  2997#, 24405
_erlst =   $29:  3051#
_erlwd =   $21:  3027#,  8354
_ermfn =   $08:  2952#, 24393
_erng  =   $23:  3033#, 23143
_ernif =   $06:  2946#
_ernob =   $25:  3039#,  5199
_ernof =   $07:  2949#
_erntl =   $26:  3042#,  7155
_ernwf =   $0a:  2958#,  6309
_erod  =   $0d:  2967#,  6240
_eroid =   $22:  3030#, 16931
_erom  =   $10:  2976#,  3715
_erov  =   $0f:  2973#, 18000
_erref =   $27:  3045#,  7518
_errg  =   $0c:  2964#,  4956
_ersn  =   $0b:  2961#, 14238
_erst  =   $19:  3003#, 13791 , 17388
_ertm  =   $16:  2994#,  6140 , 13785
_ertmf =   $01:  2931#, 23386
_eruf  =   $1b:  3009#, 16904
_erus  =   $11:  2979#,  6777
_ervfy =   $1c:  3012#, 19880
_esc   =   $1b:    34#,  8580
_ex    =   $00:   214#
_expcn =   $81:    61#,  4126 ,  7564 , 10205 , 14401 , 16806 , 16858 , 16917 , 17698
                 18124 , 18127 , 18385 , 18754 , 19174 , 19211 , 19646 , 19654 , 20677
                 20689
_fa    =   $08:   276#, 12702 , 23158 , 23392 , 23416 , 23443 , 23464 , 23521 , 23763
                 23792 , 23804 , 24326
_fn1   =   $01:   273#, 12702 , 23158 , 23170 , 23416 , 23443 , 23464 , 23521 , 23590
                 23591 , 23792 , 24153 , 24166 , 24744 , 24745 , 24754 , 24770 , 24771
                 24780 , 24781
_fn2   =   $02:   274#, 23324 , 23345 , 23792 , 24265 , 24272 , 24733 , 24754 , 24770
                 24771
_fnam1 =   $f1:   378#, 24468 , 24472 , 24481 , 24485 , 24497 , 24501 , 24505 , 24555
_fnam2 =   $f2:   384#, 24476 , 24477 , 24497 , 24501 , 24558
_fnt   =   $a5:  2526#, 14186 , 16971
_fords =   $01:   490#,  6332
_fort  =   $81:  2476#,  4251 ,  4284 ,  6303 ,  7731 ,  7842 , 23018
```

```
_get    =   $40:   122#,  5870
_gosut  =   $8d:  2494#,  4950 ,  5255 ,  6810 ,  6998
_got    =   $cb:  2577#,  3270 ,  3444 ,  7293
_gotot  =   $89:  2487#,  5045 ,  5258 ,  6996
_graph  = $2400:    63#, 22864 , 22953 , 22957 , 22961 , 22973 , 22977 , 23092 , 23096
                 23100 , 23120 , 23128
_great  =   $b1:  2545#, 13843 , 13846 , 13887
_hecmt  =   $26:  2690#,  3337 ,  4731
_hedln  =   $04:    56#,  1685 ,  3874 ,  3991 ,  4084 ,  4169 ,  4911 , 23251 , 23304
_hefnt  =   $0a:  2707#,  3539 ,  4755
_hofnt  =   $d4:  2590#,  3542
_home   =   $13:    31#, 12679 , 13477
_id     =   $d0:   414#, 24485 , 24543
_input  =   $00:   121#,  5976
_instt  =   $d4:  2589#,  3452 ,  3471
_keybd  =   $00:   128#,  1658 ,  3777 ,  5900
_keyt   =   $f9:  2633#,  5843 ,  6008
_la     =   $04:   275#, 23392 , 23400 , 24025 , 24034 , 24733 , 24754 , 24762 , 24780
                 24781
_lecmt  =   $02:  2649#,  3334 ,  3340 ,  4728 ,  4734 ,  4758
_lefnt  =   $02:  2697#,  3536 ,  4752
_leftt  =   $c8:  2572#,  3458
_lesst  =   $b3:  2549#
_lf     =   $0a:    17#,  3425 ,  5712
_lnfor  =   $12:   489#,  4275 ,  4283 ,  6286 ,  7737 ,  7751 , 23026
_lngos  =   $05:   494#,  4287 ,  4964 ,  6788 ,  6793 ,  8205 ,  8212 ,  8336 , 23033
_lnkdy  = $0101:    53#,  4064
_loopt  =   $ec:  2619#,  8252
_m1     =   $04:   222#, 17820 , 17821 , 18011 , 18015$, 18052$, 18286 , 18325 , 18485
                 18590 , 18637 , 18673
_m2     =   $03:   220#, 17824 , 17825 , 18014 , 18018$, 18051$
_m3     =   $02:   218#, 17828 , 17829 , 18017 , 18021$, 18050$
_m4     =   $01:   216#, 17800$, 17832 , 17833 , 18020 , 18024$, 18039$, 18042$,
                 18044$, 18045$
_middt  =   $ca:  2575#,  3278 ,  3455
_minut  =   $ab:  2538#,  7953 ,  7972 , 11224 , 14129 , 18966 , 22793
_monit  =   $fa:  2635#
_nextt  =   $82:  2478#,  8059
_not    = $0100:    43#
_nott   =   $a8:  2532#, 14161
_numer  =   $00:   107#, 14086
_offt   =   $24:  2687#,  8445
_onefn  =   $b4:  2551#,  3506 , 13846 , 14194
_ont    =   $91:  2500#,  8438 , 23967 , 24185 , 24223 , 24286
_ovwr   =   $f0:   419#, 24472 , 24552
_pgmbk  =   $00:    73#,  6111 , 14095 , 20026 , 23192 , 23358 , 23425 , 23455 , 23497
                 23548 , 23833
_pi     =   $ff:    41#,  2251 ,  4657 , 14109
_plust  =   $aa:  2536#, 11221 , 13887 , 14132 , 18972 , 22790
_point  =   $0a:  2706#,  3525
_print  =   $99:  2244 ,  2509#
_rcl    =   $40:   279#, 23345 , 24047 , 24074 , 24733 , 24754 , 24762
_read   =   $98:   123#,  5971
_recln  =   $e0:   407#, 24472 , 24509 , 24561
_remt   =   $8f:  2317 ,  2497#,  5169
_restt  =   $8c:  2492#,  7000
_resut  =   $d6:  2594#,  7001
_rgrt   =   $cc:  2579#
```

```
_rlrh  =  $e2:   409#, 24509 , 24546
_rows  =  $19:   345#,  9737 ,  9946
_rpl   =  $80:   280#, 23345 , 23416 , 23464 , 24367 , 24371 , 24733 , 24754 , 24762
_rs    =  $02:   130#
_runt  =  $8a:  2489#,  6997
_rvsof =  $92:    29#,  9847
_rvson =  $12:    28#,  6696 ,  9826
_sa    =  $c2:   404#, 24509 , 24540
_scrat =  $a2:  2520#
_scrn  =  $03:   131#, 20047 , 20079
_seqwr =  $e1:   418#, 24472 , 24549
_sftcr =  $8d:    20#,  8543 ,  8580 , 13476
_sg    =  $05:   224#, 18692 , 18931 , 19663 , 19730
_skip1 =  $24:    48#, 13812 , 24725
_skip2 =  $2c:    49#,  3576 ,  3710 ,  4772 ,  5011 ,  5814 ,  5820 ,  5972 ,  6450
                 6562 ,  8035 ,  8349 ,  9205 ,  9403 ,  9431 , 11674 , 11690 , 13146
                13155 , 13189 , 13233 , 13786 , 14211 , 14217 , 15142 , 16899 , 18381
                18621 , 19852 , 21375 , 21703 , 23435 , 24394 , 24400
_spc   =  $20:    39#,  1979 ,  2049 ,  4611 ,  5813 ,  7448 , 11301 , 12789 , 15626
                15747 , 15766 , 19132 , 19188 , 20855 , 20933 , 23277
_spct  =  $a6:  2528#,  5669
_stadr =  $02:   285#, 12703 , 23465 , 23473 , 23474 , 23522 , 24118 , 24126
_step  =  $03:   491#,  6320
_stept =  $a9:  2534#,  7808
_stpky =  $03:    16#, 13476
_stpsg =  $08:   492#,  6328
_strng =  $ff:   108#
_sysbk =  $0f:    75#,  1758
_tabt  =  $a3:  2522#,  3286 ,  3292 ,  5666
_talk  =  $60:   402#, 23196 , 23370 , 23372 , 23703 , 23867 , 24823
_thent =  $a7:  2530#,  5048 ,  6999
_tidln =  $06:    57#,  5363 ,  5397
_tkofs =  $80:    51#,  3298 ,  4734 ,  4758
_tmpst =  $1b:    59#, 20264 , 20437
_tot   =  $a4:  2524#,  6820 ,  7302 ,  7771 ,  9687 ,  9704 , 24195
_trapt =  $d7:  2596#,  7002
_untlt =  $fc:  2639#,  8175 ,  8317
_upgrp =  $8e:    23#,  9923
_uplo  =  $0e:    22#,  9914
_usint =  $fb:  2637#,  3281 ,  5658
_varbk =  $01:    74#,  8413 , 15497 , 15594 , 20027
_whilt =  $fd:  2641#,  8178 ,  8314
```

Spitzen-Software für Commodore 128/128 D

WordStar 3.0 mit MailMerge

Der Bestseller unter den Textverarbeitungsprogrammen für PCs bietet Ihnen bildschirmorientierte Formatierung, deutschen Zeichensatz und DIN-Tastatur sowie integrierte Hilfstexte. Mit MailMerge können Sie Serienbriefe mit persönlicher Anrede an eine beliebige Anzahl von Adressen schreiben und auch die Adreßaufkleber drucken.

WordStar/MailMerge für den Commodore 128 PC
Bestell-Nr. MS 103 (5¼"-Diskette)

Hardware-Anforderungen: Commodore 128 PC, Diskettenlaufwerk, 80-Zeichen-Monitor, beliebiger Commodore-Drucker oder ein Drucker mit Centronics-Schnittstelle

Für nur DM 199,–* (sFr. 178,–/öS 1890,–*)

***inkl. MwSt. Unverbindliche Preisempfehlung**

Markt&Technik
128er-Software

MicroPro
WordStar 3.0
mit MailMerge für den Commodore 128 PC

5¼"-Diskette
im Floppy 1541-Format

Und dazu die weiterführende Literatur:

Mit diesem Buch haben Sie eine wertvolle Ergänzung zum WordStar-Handbuch: Anhand vieler Beispiele steigen Sie mühelos in die Praxis der Textverarbeitung mit **WordStar** ein. Angefangen beim einfachen Brief bis hin zur umfangreichen Manuskripterstellung zeigt Ihnen dieses Buch auch, wie Sie mit Hilfe von MailMerge Serienbriefe an eine beliebige Anzahl von Adressen mit persönlicher Anrede senden können.

Best.-Nr. MT 780
ISBN 3-89090-181-6
DM 49,– (sFr. 45,10/öS 382,20)

Erhältlich bei Ihrem Buchhändler.

Sie erhalten jedes **WordStar-Programm** für Ihren Commodore 128 fertig angepaßt (Bildschirmsteuerung). **Jeweils Originalprodukte!** Jedes Programmpaket enthält außerdem ein ausführliches Handbuch mit kompakter Befehlsübersicht.

Diese Markt&Technik-Softwareprodukte erhalten Sie in den Computer-Abteilungen der Kaufhäuser oder bei Ihrem Computerhändler.

Markt&Technik

Unternehmensbereich Buchverlag

Hans-Pinsel-Straße 2, 8013 Haar bei München

Spitzen-Software für Commodore 128/128 D

dBASE II, Version 2.41

dBASE II, das meistverkaufte Programm unter den Datenbanksystemen, eröffnet Ihnen optimale Möglichkeiten der Daten- u. Dateihandhabung. Einfach u. schnell können Datenstrukturen definiert, benutzt und geändert werden. Der Datenzugriff erfolgt sequentiell oder nach frei wählbaren Kriterien, die integrierte Kommandosprache ermöglicht den Aufbau kompletter Anwendungen wie Finanzbuchhaltung, Lagerverwaltung, Betriebsabrechnung usw.

dBASE II für den Commodore 128 PC
Bestell-Nr. MS 303 (5¼"-Diskette)

Hardware-Anforderungen: Commodore 128 PC, Diskettenlaufwerk, 80-Zeichen-Monitor, beliebiger Commodore-Drucker oder ein Drucker mit Centronics-Schnittstelle

Für nur DM 199,–* (sFr. 178,–/öS 1890,–*)

***inkl. MwSt. Unverbindliche Preisempfehlung**

Markt&Technik
128er-Software

dBASE™ II

ASHTON-TATE
für den
Commodore 128 PC

5¼"-Diskette
im Floppy 1541-Format

Und dazu die weiterführende Literatur:

Zu einem Weltbestseller unter den Datenbanksystemen gehört auch ein klassisches Einführungs- und Nachschlagewerk! Dieses Buch von dem deutschen Erfolgsautor Dr. Peter Albrecht begleitet Sie mit nützlichen Hinweisen bei Ihrer täglichen Arbeit mit **dBASE II.** Schon nach Beherrschung weniger Befehle ist der Einsteiger in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten.

Best.-Nr. MT 838
ISBN 3-89090-189-1
DM 49,– (sFr. 45,10/öS 382,20)

Erhältlich bei Ihrem Buchhändler.

Sie erhalten jedes **dBASE II-Programm** für Ihren Commodore 128 PC fertig angepaßt (Bildschirmsteuerung). **Jeweils Originalprodukte!** Jedes Programmpaket enthält außerdem ein ausführliches Handbuch mit kompakter Befehlsübersicht.

Diese Markt&Technik-Softwareprodukte erhalten Sie in den Computer-Abteilungen der Kaufhäuser oder bei Ihrem Computerhändler

Markt&Technik

Unternehmensbereich Buchverlag

Hans-Pinsel-Straße 2, 8013 Haar bei München

Spitzen-Software für Commodore 128/128 D

MULTIPLAN, Version 1.06

Wenn Sie die zeitraubende manuelle Verwaltung tabellarischer Aufstellungen mit Bleistift, Radiergummi und Rechenmaschine satt haben, dann ist MULTIPLAN, das System zur Bearbeitung »elektronischer Datenblätter«, genau das richtige für Sie! Das benutzerfreundliche und leistungsfähige Tabellenkalkulationsprogramm kann bei allen Analyse- und Planungsberechnungen eingesetzt werden wie z.B. Budgetplanungen, Produktkalkulationen, Personalkosten usw. Spezielle Formatierungs-, Aufbereitungs- und Druckanweisungen ermöglichen außerdem optimal aufbereitete Präsentationsunterlagen!

MULTIPLAN für den Commodore 128 PC
Bestell-Nr. MS 203 (5¼"-Diskette)

Hardware-Anforderungen: Commodore 128 PC, Diskettenlaufwerk, 80-Zeichen-Monitor, beliebiger Commodore-Drucker oder ein Drucker mit Centronics-Schnittstelle

Für nur DM 199,–* (sFr. 178,–/öS 1890,–*)

***inkl. MwSt. Unverbindliche Preisempfehlung**

Markt&Technik
128er-Software

MICROSOFT
MULTIPLAN
für den
Commodore 128 PC

5¼"-Diskette
im Floppy 1541-Format

Und dazu die weiterführende Literatur:

Dank seiner Menütechnik ist **MULTIPLAN** sehr schnell erlernbar. Mit diesem Buch von Dr. Peter Albrecht werden Sie Ihre Tabellenkalkulation ohne Probleme in den Griff bekommen. Als Nachschlagewerk leistet es auch dem Profi nützliche Dienste.

Best.-Nr. MT 836
ISBN 3-89090-187-5
DM 49,– (sFr. 45,10/öS 382,20)

Erhältlich bei Ihrem Buchhändler.

Sie erhalten jedes **MULTIPLAN-Programm** für Ihren Commodore 128 PC fertig angepaßt (Bildschirmsteuerung). **Jeweils Originalprodukte!** Jedes Programmpaket enthält außerdem ein ausführliches Handbuch mit kompakter Befehlsübersicht.

Diese Markt&Technik-Softwareprodukte erhalten Sie in den Computer-Abteilungen der Kaufhäuser oder bei Ihrem Computerhändler.

Markt&Technik

Unternehmensbereich Buchverlag

Hans-Pinsel-Straße 2, 8013 Haar bei München

TOP-ASS

Der ASE-Macro-Assembler für den Commodore 128 PC mit integriertem Editor, Monitor und Linker.

Dieser 6502-Macroassembler setzt neue Maßstäbe. Seine Leistungsfähigkeit wird jeden verwöhnten Maschinenprogrammierer überzeugen:

- integrierter Editor, der schon bei der Eingabe des Quelltextes eine Syntaxüberprüfung vornimmt;
- integrierter Linker, mit dem quellgesteuertes Linken von relokatiblen Modulen möglich ist;
- assemblereigene schnelle und gleichzeitig sehr leistungsfähige Integerarithmetik;
- über 2000 Labels können gleichzeitig verwaltet werden, das heißt Maschinenprogramme bis zu einer Länge von ca. 25 KByte Objektcode können bei Bedarf in einem Rutsch assembliert werden;
- Macros mit beliebig vielen Parametern, Macrobibliotheken, Minimacs, bedingte Assemblierung, Labeleingabe im Dialog, Ausgabe formatierter Assemblerlistings, Ausgabe sortierter Symboltabellen und vieles andere mehr.

Außerdem wird der ASE-Macroassembler von einem sehr guten Monitor und einem Relativlader unterstützt, der relokatible Module an beliebige Speicheradressen laden kann und endlich Schluß macht mit den Dutzenden Maschinenprogrammen auf Diskette, die sich nur durch ihre Startadresse unterscheiden!

Lernen Sie es kennen, das TOP-ASS Assembler-Entwicklungssystem! Es lohnt sich!

Best.-Nr. MD 253A **DM 89,–**
inkl. MwSt. Unverbindliche Preisempfehlung.

PROTEXT

Die Profi-Textverarbeitung mit vollautomatischer Silbentrennung, integrierter Tabellenkalkulation und Zusatzprogramm zum Überprüfen der Rechtschreibung für den Commodore 128 PC.

PROTEXT ist ein leicht bedienbares Textprogramm mit hoher Leistungsfähigkeit. Eingebaute Hilfefunktionen ermöglichen eine schnelle Einarbeitung. Mit **PROTEXT** sind daher auch Anfänger in der Lage, alle Vorteile eines professionellen Textprogramms zu nutzen. Überzeugen Sie sich selbst!

Was **PROTEXT** alles kann:

- Farbkombination für Hintergrund und Schrift (Vordergrund) frei wählbar;
- formatierte Ausgabe auf Bildschirm und Drucker mit programmierbaren Haltepunkten über serielle, V24- oder zwei Software-Centronics-Schnittstellen;
- vielfältige Formatanweisungen: linker/rechter Rand, vollautomatische Silbentrennung, Kopf-/Fußzeilen, Fußnoten, Zentrieren usw.;
- schnelle selbstlernende Textkorrektur mit deutschem (ca. 25000 Worte) Grundwortschatz sowie neun Kundenbibliotheken, die in Text umgewandelt, bearbeitet, ergänzt, sortiert und ausdruckbar sind;
- Textübertragung per DFÜ mit Space-Optimierung und automatischer Fehlerkorrektur;
- leistungsfähige Rechenmöglichkeiten mit Zeilenmarkierung (Rechentabulator), Kolonnenverarbeitung, progr. Tabellenkalkulation und Taschenrechner.

Hardwareanforderungen: C128 oder C128D, 80-Zeichen-Monitor, Commodore-Drucker oder Drucker mit Centronics-Schnittstelle.

Best.-Nr. MD 254A **DM 89,–** inkl. MwSt. Unverbindliche Preisempfehlung.

TOP-ASS und PROTEXT erhalten Sie in den Computer-Abteilungen der Kaufhäuser und in Computershops.

Markt&Technik
Unternehmensbereich Buchverlag
Hans-Pinsel-Straße 2, 8013 Haar bei München